Honeypots That Bite Back: A Fuzzy Technique for Identifying and Inhibiting Fingerprinting Attacks on Low Interaction Honeypots

Nitin Naik[1], Paul Jenkins[1], Roger Cooke[1] and Longzhi Yang[2]

[1]Defence School of Communications and Information Systems, Ministry of Defence, United Kingdom
[2]Department of Computer and Information Sciences, Northumbria University, United Kingdom
Email: {nitin.naik100, paul.jenkins683, roger.cooke472}@mod.gov.uk, longzhi.yang@northumbria.ac.uk

***Abstract*—The development of a robust strategy for network security is reliant upon a combination of in-house expertise and for completeness attack vectors used by attackers. A honeypot is one of the most popular mechanisms used to gather information about attacks and attackers. However, low-interaction honeypots only emulate an operating system and services, and are more prone to a fingerprinting attack, resulting in severe consequences such as revealing the identity of the honeypot and thus ending the usefulness of the honeypot forever, or worse, enabling it to be converted into a bot used to attack others. A number of tools and techniques are available both to fingerprint low-interaction honeypots and to defend against such fingerprinting; however, there is an absence of fingerprinting techniques to identify the characteristics and behaviours that indicate fingerprinting is occurring. Therefore, this paper proposes a fuzzy technique to correlate the attack actions and predict the probability that an attack is a fingerprinting attack on the honeypot. Initially, an experimental assessment of the fingerprinting attack on the low-interaction honeypot is performed, and a fingerprinting detection mechanism is proposed that includes the underlying principles of popular fingerprinting attack tools. This implementation is based on a popular and commercially available low-interaction honeypot for Windows - KFSensor. However, the proposed fuzzy technique is a general technique and can be used with any low-interaction honeypot to aid in the identification of the fingerprinting attack whilst it is occurring; thus protecting the honeypot from the fingerprinting attack and extending its life.**

I. Introduction

The Honeypot is one of the most popular mechanisms to gather information about attacks and attackers, and is deliberately placed on the organisational network to be probed and attacked. A honeypot deceives and attracts an attacker who attempts to gain unauthorized access to the network [1]. As a result, the honeypot can gain information about the attacker, their attack tools and techniques, or it can divert an attacker from the real targets [2]. Honeypots are generally classified as high-interaction or low-interaction. High-interaction honeypots are very expensive and have higher probability of being hijacked by the attacker [3]. Conversely, low-interaction honeypots are inexpensive and have a lower likelihood of being hijacked by the attacker [3]. However, the latter emulates an operating system and other services, and has a greater probability of being attacked by a fingerprinting or identity probing attack, to collect the complete details of the honeypot. In general, fingerprinting is relatively harmless, as it is intrinsically used to detail the system, however, it has severe consequences for the honeypot as it can reveal the identity of the honeypot and may end the usefulness of the honeypot.

A fingerprinting attack requires a sequence of actions by the attacker to collect fingerprints of the target system. For example, in OS fingerprinting, an attacker has to examine several TCP fields, Timestamps, Window Scaling, Maximum Segment Size, Explicit Congestion Notification and IP Identification together to reveal the fingerprint [4]. A number of tools and techniques are available to fingerprint low-interaction honeypots and to defend against such fingerprinting; however, the issue is to attempt to identify fingerprinting in real-time and in situ. There is an absence of fingerprinting techniques to identify the characteristics and behaviours that indicate fingerprinting is occurring. Almost all low-interaction honeypots lack intelligence to indicate that fingerprinting is occurring. The principal difficulty in identifying that such an attack is happening is: to correctly correlate all attack actions and differentiate between an individual attack and a fingerprinting attack. Despite the possibility of errors in their correlation, fuzzy logic has the ability to correlate all attack actions in a given range successfully to identify a fingerprinting attack [5], [6], [7], [8], [9], [10], [11], [12]. Using this feature of fuzzy logic, this paper proposes a fuzzy technique to correlate the attack actions and identify that an attack is a fingerprinting attack [13], [14].

Firstly, the paper presents an experimental assessment of the fingerprinting attack on the low-interaction honeypot - KFSensor. The KFSensor honeypot was selected for this experiment because it is a popular and commercially available low-interaction honeypot for Windows and which is actively maintained. Subsequently, based on this experimental assessment, the paper proposes a fingerprinting detection mechanism for the low-interaction honeypot, which includes the underlying principles of popular fingerprinting attack tools. Finally, based on the proposed fingerprinting detection mechanism, a fuzzy technique for detecting the fingerprinting attack on low-interaction honeypots is proposed. This fuzzy technique is a general technique and can be used with any low-interaction honeypots to aid in the identification of the fingerprinting attack whilst it is occurring; thus this can protect the honeypot from the fingerprinting attack and extend its life.

The remaining paper is organised as follows. Section II presents the background information on honeypots and the fingerprinting attack. Section III presents the experimental assessment of the fingerprinting attack on the low-interaction honeypot, KFSensor. Section IV proposes a detection mechanism for the fingerprinting attack on the low-interaction honeypot. Section V proposes the fuzzy technique for detecting the fingerprinting attack on low-interaction honeypots. Section VI presents the test results of the simulated attacks on low-interaction honeypots and the success of the proposed fuzzy technique. Finally, Section VII presents the conclusion and specifies possible future extensions of the proposed fuzzy technique for the low-interaction honeypot.

II. Background Information

A. Honeypots and Honeypot Types

A honeypot is a security system designed for deceiving, detecting and diverting an attacker, whilst simultaneously collecting information to prevent the honeypot being utilised in an attack [1]. Residing on the network, it remains idle until triggered, causing a reaction that will produce information allowing the identification of the attacker [2]. The honeypot is monitored continuously by in-house security experts to reveal the vulnerabilities of their network while evolving an improved defensive strategy [15]. Honeypots can be grouped into two major categories, low-interaction honeypots and high-interaction honeypots, based on the level of activity they allow an attacker to perform. Low-interaction honeypots commonly attempt to limit the interaction capability with an attacker by emulating services and an operating system [3]. High-interaction honeypots are complex systems that have an unrestricted interaction capability with an attacker, offering a suite of real services and a real operating system, nothing is emulated [3]. Obviously, honeypots should not be used as a defensive mechanism for the network they reside in.

B. Fingerprinting Attack

Fingerprinting is a technique to collect information about a remote system for the purpose of its identification. It can be an active operation where carefully crafted packets are sent to the target system to analyse its response, or a passive operation where the network traffic from the target system is analysed. Operating System (OS) fingerprinting is the most common type of fingerprinting, which works by sending carefully crafted packets to various open and closed ports on the target system and receiving responses containing fingerprint information [4]. Every OS replies differently to the same query and the difference between the responses generated by two OS's reveals significant information to the attacker. Moreover, every OS implements a TCP/IP stack differently which in turn results in different responses and this enables the OS fingerprinting attack to become successful.

III. Experimental Assessment of the Fingerprinting Attack on the Low-Interaction Honeypot - KFSensor

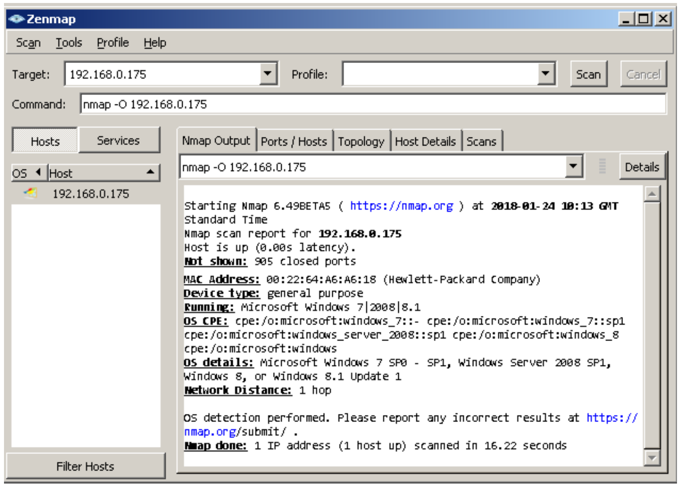
To demonstrate the fingerprinting attack on the low-interaction honeypot KFSensor, simulation of the fingerprinting attack is carried out using the Nmap scanning tool, allowing post-experiment analysis of the results to discover various indicators of the fingerprinting attack. Nmap is an effective tool for use in the fingerprinting attack and is capable of performing *fuzzy* fingerprinting when it cannot find a perfect fingerprint match. While performing OS fingerprinting, Nmap sends up to 16 TCP, UDP, and ICMP probes (excluding retransmission) to known open and closed ports of the target system [16]. All these probes are designed to exploit various ambiguities present in the standard protocol RFCs. Upon receiving responses from the target system, it analyses several attributes of those responses and combines them to generate a fingerprint. Every probe packet is tracked and if no response is received, it retransmits that probe packet at least once. These probe packets are IPv4 packets with random IP ID values. Probes sent to open TCP ports are skipped when there is no open port found, while a probe is sent to closed TCP and UDP ports and if no closed port is found, Nmap selects another port at random and retries [17].

The experiments on the fingerprinting attack utilise five different fingerprinting attack scripts with various sub-options as shown in Figs. 1 to 5, which are run to perform the OS fingerprinting attack on the KFSensor to obtaining details of OS and devices of the system that are running on the KFSensor honeypot. The first fingerprinting attack script in Fig. 1 detects OS fingerprints of the target machine and presents a detailed description for each fingerprint [18]. This information contains the device type and the OS details in the format of free-form data. The fingerprint may include a perfect match against OS version numbers, device models, and architectures specific to a given fingerprint [19]. The second fingerprinting attack script in Fig. 2 performs OS detection, version detection, script scanning, and traceroute. The third fingerprinting attack script in Fig. 3 shows a *fuzzy* technique for a powerful fingerprinting attack, performing aggressive fingerprinting and OS guesses, which provides very close OS match results when it could not find the perfect OS match [20]. Nmap displays all the near-matches with their confidence level in percentages. The fourth fingerprinting attack script in Fig. 4 attempts to find the version of the service (e.g., HTTP, SSH, FTP, Telnet, SMTP, DNS) running on the port, where, a higher intensity level (0 to 9) increases the probability of the correctness of the fingerprinting results [21], [22]. The fifth and final fingerprinting attack script in Fig. 5 performs remote OS detection by trying several times as shown in the script.

The traffic analysis is conducted on the two separate logs captured by the KFSensor honeypot and Wireshark analyser for comparative analysis purposes. Wireshark offers detailed analysis of traffic and is used here mainly to capture the entire traffic as precaution that KFSensor could not, possibly due to configuration issues and has prioritised the events for log recording.

IV. Proposed Detection Mechanism for the Fingerprinting Attack on the Low-Interaction Honeypot

The proposed detection mechanism for the fingerprinting attack is based on the assessment of the previous experiments and the underlying attack principles of popular fingerprinting tools such as Nmap, Xprobe2, HTTPrint, IPlog, Amap and Nessus. Most of these tools utilise similar principles in their fingerprinting technique; which can easily be derived from



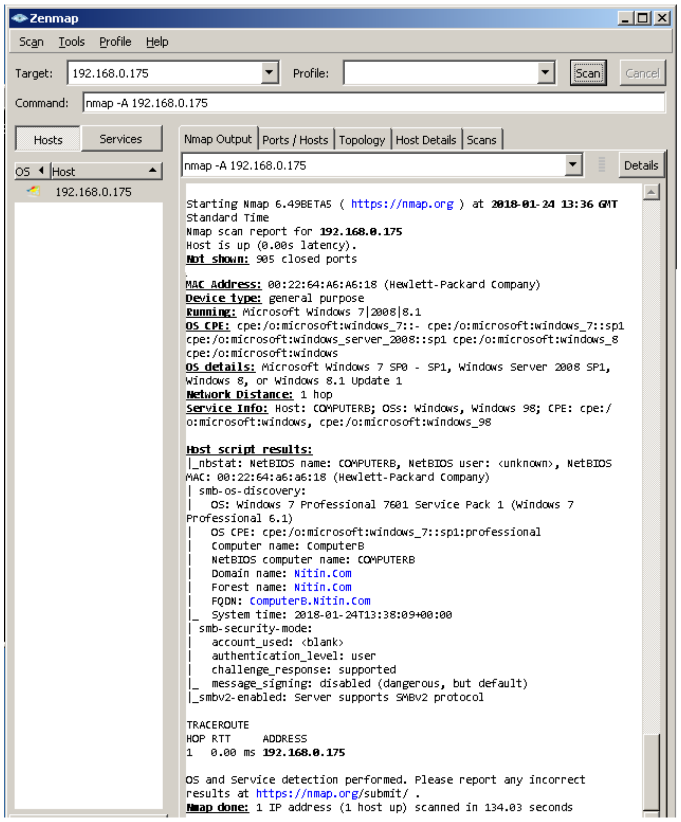
Target: 192.168.0.175

Command: nmap -O 192.168.0.175

192.168.0.175

```
Starting Nmap 6.49BETA5 ( https://nmap.org ) at 2018-01-24 10:13 GMT
Standard Time
Nmap scan report for 192.168.0.175
Host is up (0.00s latency).
Not shown: 905 closed ports
MAC Address: 00:22:64:A6:A6:18 (Hewlett-Packard Company)
Device type: general purpose
Running: Microsoft Windows 7|2008|8.1
OS CPE: cpe:/o:microsoft:windows_7::- cpe:/o:microsoft:windows_7::sp1
cpe:/o:microsoft:windows_server_2008::sp1 cpe:/o:microsoft:windows_8
cpe:/o:microsoft:windows
OS details: Microsoft Windows 7 SP0 - SP1, Windows Server 2008 SP1,
Windows 8, or Windows 8.1 Update 1
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at https://
nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 16.22 seconds
```

Fig. 1. Fingerprinting attack script for remote OS detection using TCP/IP stack



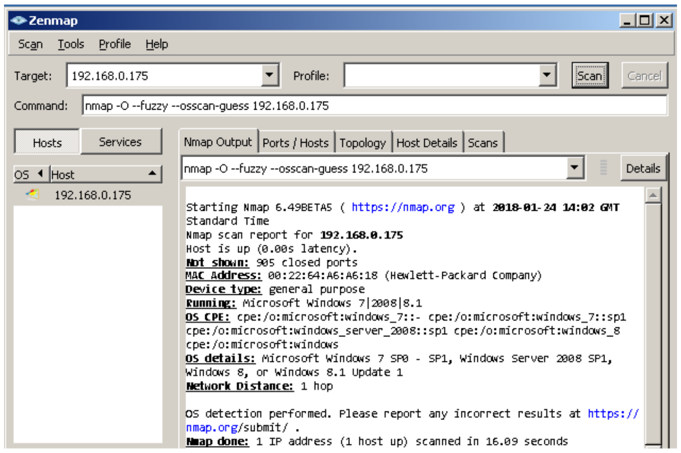
Target: 192.168.0.175

Command: nmap -A 192.168.0.175

192.168.0.175

```
Starting Nmap 6.49BETA5 ( https://nmap.org ) at 2018-01-24 13:36 GMT
Standard Time
Nmap scan report for 192.168.0.175
Host is up (0.00s latency).
Not shown: 905 closed ports

MAC Address: 00:22:64:A6:A6:18 (Hewlett-Packard Company)
Device type: general purpose
Running: Microsoft Windows 7|2008|8.1
OS CPE: cpe:/o:microsoft:windows_7::- cpe:/o:microsoft:windows_7::sp1
cpe:/o:microsoft:windows_server_2008::sp1 cpe:/o:microsoft:windows_8
cpe:/o:microsoft:windows
OS details: Microsoft Windows 7 SP0 - SP1, Windows Server 2008 SP1,
Windows 8, or Windows 8.1 Update 1
Network Distance: 1 hop
Service Info: Host: COMPUTERB; OSs: Windows, Windows 98; CPE: cpe:/
o:microsoft:windows, cpe:/o:microsoft:windows_98

Host script results:
|_nbstat: NetBIOS name: COMPUTERB, NetBIOS user: <unknown>, NetBIOS
MAC: 00:22:64:a6:a6:18 (Hewlett-Packard Company)
| smb-os-discovery:
|   OS: Windows 7 Professional 7601 Service Pack 1 (Windows 7
Professional 6.1)
|   OS CPE: cpe:/o:microsoft:windows_7::sp1:professional
|   Computer name: ComputerB
|   NetBIOS computer name: COMPUTERB
|   Domain name: Nitin.Com
|   Forest name: Nitin.Com
|   FQDN: ComputerB.Nitin.Com
|_  System time: 2018-01-24T13:38:09+00:00
| smb-security-mode:
|   account_used: <blank>
|   authentication_level: user
|   challenge_response: supported
|_  message_signing: disabled (dangerous, but default)
|_smbv2-enabled: Server supports SMBv2 protocol

TRACEROUTE
HOP RTT     ADDRESS
1   0.00 ms 192.168.0.175

OS and Service detection performed. Please report any incorrect
results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 134.03 seconds
```

Fig. 2. Fingerprinting attack script for OS detection, version detection, script scanning and traceroute



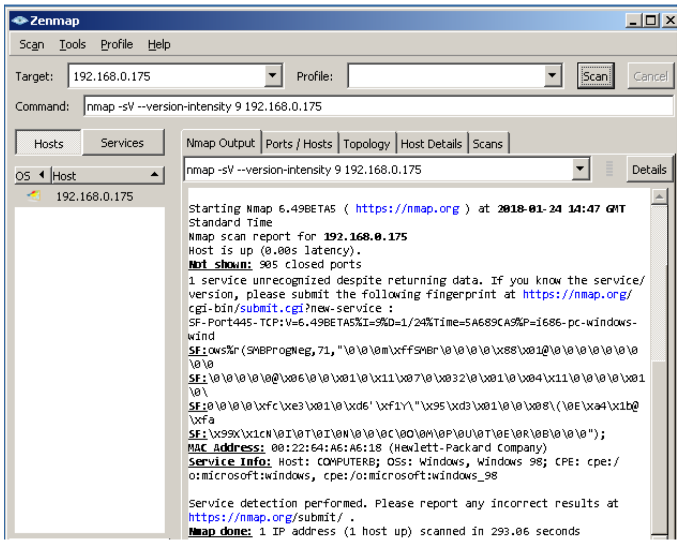
Target: 192.168.0.175

Command: nmap -O --fuzzy --osscan-guess 192.168.0.175

192.168.0.175

```
Starting Nmap 6.49BETA5 ( https://nmap.org ) at 2018-01-24 14:02 GMT
Standard Time
Nmap scan report for 192.168.0.175
Host is up (0.00s latency).
Not shown: 905 closed ports
MAC Address: 00:22:64:A6:A6:18 (Hewlett-Packard Company)
Device type: general purpose
Running: Microsoft Windows 7|2008|8.1
OS CPE: cpe:/o:microsoft:windows_7::- cpe:/o:microsoft:windows_7::sp1
cpe:/o:microsoft:windows_server_2008::sp1 cpe:/o:microsoft:windows_8
cpe:/o:microsoft:windows
OS details: Microsoft Windows 7 SP0 - SP1, Windows Server 2008 SP1,
Windows 8, or Windows 8.1 Update 1
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at https://
nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 16.09 seconds
```

Fig. 3. Fingerprinting attack script (FUZZY) for detecting closest OS match results



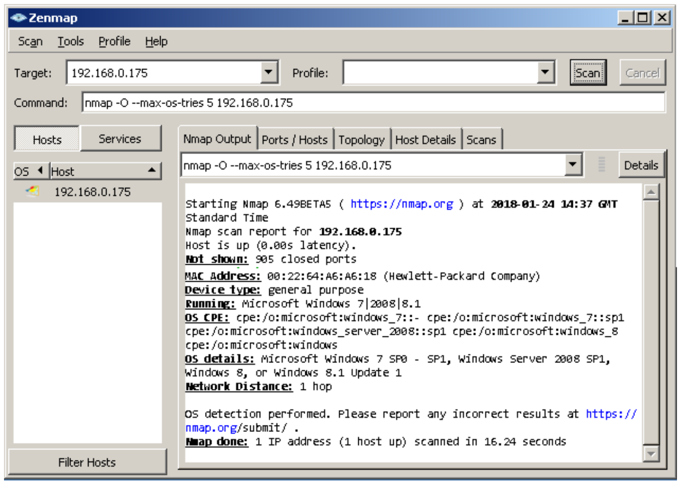
Target: 192.168.0.175

Command: nmap -sV --version-intensity 9 192.168.0.175

192.168.0.175

```
Starting Nmap 6.49BETA5 ( https://nmap.org ) at 2018-01-24 14:47 GMT
Standard Time
Nmap scan report for 192.168.0.175
Host is up (0.00s latency).
Not shown: 905 closed ports
1 service unrecognized despite returning data. If you know the service/
version, please submit the following fingerprint at https://nmap.org/
cgi-bin/submit.cgi?new-service :
SF-Port445-TCP:V=6.49BETA5%I=9%D=1/24%Time=5A689CA9%P=i686-pc-windows-
wind
SF:ows%r(SMBProgNeg,71,"\0\0\0m\xffSMBr\0\0\0\0\x88\x01@\0\0\0\0\0\0\0
\0\0
SF:\0\0\0\0\0@\x06\0\0\x01\0\x11\x07\0\x032\0\x01\0\x04\x11\0\0\0\0\x01
\0\
SF:0\0\0\0\xfc\xe3\x01\0\xd6'\xf1Y\"\x95\xd3\x01\0\0\x08\(\0E\xa4\x1b@
\xfa
SF:\x99X\x1cN\0I\0T\0I\0N\0\0\0C\0O\0M\0P\0U\0T\0E\0R\0B\0\0\0");
MAC Address: 00:22:64:A6:A6:18 (Hewlett-Packard Company)
Service Info: Host: COMPUTERB; OSs: Windows, Windows 98; CPE: cpe:/
o:microsoft:windows, cpe:/o:microsoft:windows_98

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 293.06 seconds
```

Fig. 4. Fingerprinting attack script for determining the version of the service running on the port



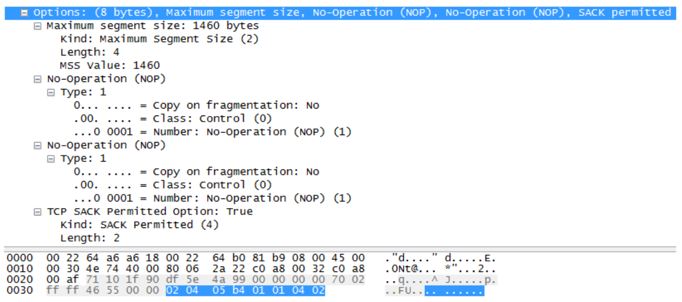
Target: 192.168.0.175

Command: nmap -O --max-os-tries 5 192.168.0.175

192.168.0.175

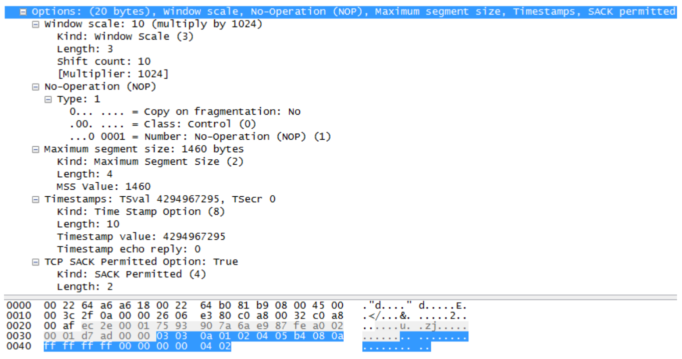
```
Starting Nmap 6.49BETA5 ( https://nmap.org ) at 2018-01-24 14:37 GMT
Standard Time
Nmap scan report for 192.168.0.175
Host is up (0.00s latency).
Not shown: 905 closed ports
MAC Address: 00:22:64:A6:A6:18 (Hewlett-Packard Company)
Device type: general purpose
Running: Microsoft Windows 7|2008|8.1
OS CPE: cpe:/o:microsoft:windows_7::- cpe:/o:microsoft:windows_7::sp1
cpe:/o:microsoft:windows_server_2008::sp1 cpe:/o:microsoft:windows_8
cpe:/o:microsoft:windows
OS details: Microsoft Windows 7 SP0 - SP1, Windows Server 2008 SP1,
Windows 8, or Windows 8.1 Update 1
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at https://
nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 16.24 seconds
```

Fig. 5. Fingerprinting attack script for for remote OS detection with the given number attempts



```
⊟ Options: (8 bytes), Maximum segment size, No-Operation (NOP), No-Operation (NOP), SACK permitted
  ⊟ Maximum segment size: 1460 bytes
      Kind: Maximum Segment Size (2)
      Length: 4
      MSS Value: 1460
  ⊟ No-Operation (NOP)
    ⊟ Type: 1
        0... .... = Copy on fragmentation: No
        .00. .... = Class: Control (0)
        ...0 0001 = Number: No-Operation (NOP) (1)
  ⊟ No-Operation (NOP)
    ⊟ Type: 1
        0... .... = Copy on fragmentation: No
        .00. .... = Class: Control (0)
        ...0 0001 = Number: No-Operation (NOP) (1)
  ⊟ TCP SACK Permitted Option: True
      Kind: SACK Permitted (4)
      Length: 2

0000  00 22 64 a6 a6 18 00 22  64 b0 81 b9 08 00 45 00   ."d...." d.....E.
0010  00 30 4e 74 40 00 80 06  2a 22 c0 a8 00 32 c0 a8   .0Nt@... *"...2..
0020  00 af 71 10 1f 90 df 5e  4a 99 00 00 00 00 70 02   ..q....^ J.....p.
0030  ff ff 46 55 00 00 02 04  05 b4 01 01 04 02         ..FU.... ......
```

Fig. 6. Wireshark capturing TCP Options in a normal TCP packet



```
⊟ Options: (20 bytes), Window scale, No-Operation (NOP), Maximum segment size, Timestamps, SACK permitted
  ⊟ Window scale: 10 (multiply by 1024)
      Kind: Window Scale (3)
      Length: 3
      Shift count: 10
      [Multiplier: 1024]
  ⊟ No-Operation (NOP)
    ⊟ Type: 1
        0... .... = Copy on fragmentation: No
        .00. .... = Class: Control (0)
        ...0 0001 = Number: No-Operation (NOP) (1)
  ⊟ Maximum segment size: 1460 bytes
      Kind: Maximum Segment Size (2)
      Length: 4
      MSS Value: 1460
  ⊟ Timestamps: TSval 4294967295, TSecr 0
      Kind: Time Stamp Option (8)
      Length: 10
      Timestamp value: 4294967295
      Timestamp echo reply: 0
  ⊟ TCP SACK Permitted Option: True
      Kind: SACK Permitted (4)
      Length: 2

0000  00 22 64 a6 a6 18 00 22  64 b0 81 b9 08 00 45 00   ."d...." d.....E.
0010  00 3c 2f 0a 00 00 26 06  e3 80 c0 a8 00 32 c0 a8   .</...&. .....2..
0020  00 af ec 2e 00 01 75 93  90 7a 6a e9 87 fe a0 02   ......u. .zj.....
0030  00 01 d7 ad 00 00 03 03  0a 01 02 04 05 b4 08 0a   ........ ........
0040  ff ff ff ff 00 00 00 00  04 02                     ........ ..
```

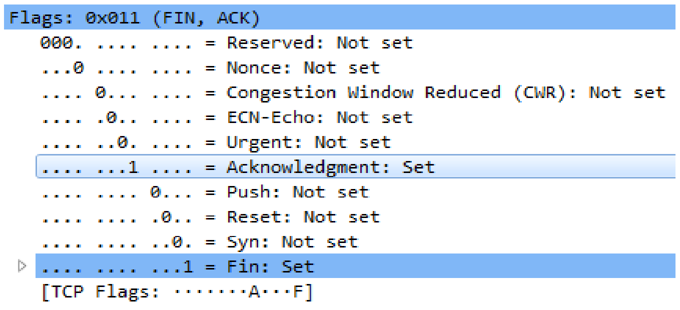
Fig. 7. Wireshark capturing TCP Options in an abnormal TCP packet

the experiments presented on fingerprinting using Nmap. In TCP/IP networks, the OS fingerprinting technique is called TCP/IP stack fingerprinting, based on the TCP, ICMP and UDP packets. Some fingerprinting tools employ TCP packets as the main weapon for the fingerprinting attack, while other tools employ ICMP and UDP packets. Therefore, to determine the most appropriate and effective detection mechanism for detecting signs of a fingerprinting attack requires a detailed investigation of TCP, ICMP and UDP packets.

A. Detection Mechanism based on the Investigation of TCP Packets

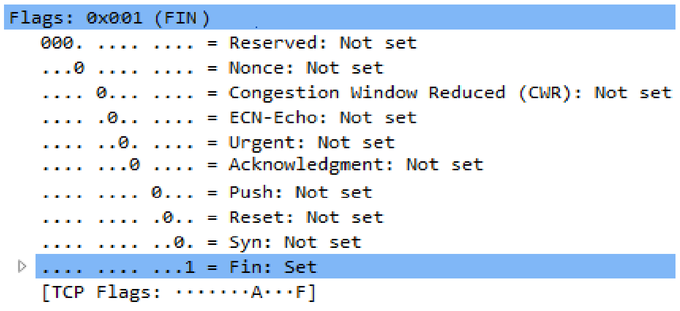
A fingerprinting attack is profoundly reliant on TCP packets and its six flags (SYN, ACK, URG, PSH, RST, FIN) which control different aspects of communications. The different combinations of six TCP flags with two additional flags (CWR, ECN) and 3 Reserved Bits are used to perform various types of probing in the fingerprinting attack.

- **TCP Options:** TCP Options is one of the most important fields of the TCP header and is utilised by the majority of fingerprinting tools due to its customised settings and variable size of 0 to 40 bytes. Its patterns of compliance for any particular OS can reveal the target OS. Interestingly, the same TCP Options can be used as a counter-measure and to locate the potential fingerprinting attacks. Here the two TCP Options are illustrated which are collected on the KFSensor machine during the experiment. Fig. 6 shows the TCP Options of a normal TCP packet and, Fig. 7 shows the TCP Options of a packet sent for the purpose of the fingerprinting attack.



```
Flags: 0x011 (FIN, ACK)
  000. .... .... = Reserved: Not set
  ...0 .... .... = Nonce: Not set
  .... 0... .... = Congestion Window Reduced (CWR): Not set
  .... .0.. .... = ECN-Echo: Not set
  .... ..0. .... = Urgent: Not set
  .... ...1 .... = Acknowledgment: Set
  .... .... 0... = Push: Not set
  .... .... .0.. = Reset: Not set
  .... .... ..0. = Syn: Not set
▷ .... .... ...1 = Fin: Set
  [TCP Flags: ·······A···F]
```

Fig. 8. Wireshark capturing normal pattern of FIN-ACK flags



```
Flags: 0x001 (FIN)
  000. .... .... = Reserved: Not set
  ...0 .... .... = Nonce: Not set
  .... 0... .... = Congestion Window Reduced (CWR): Not set
  .... .0.. .... = ECN-Echo: Not set
  .... ..0. .... = Urgent: Not set
  .... ...0 .... = Acknowledgment: Not set
  .... .... 0... = Push: Not set
  .... .... .0.. = Reset: Not set
  .... .... ..0. = Syn: Not set
▷ .... .... ...1 = Fin: Set
  [TCP Flags: ·······A···F]
```

Fig. 9. Wireshark capturing abnormal pattern of FIN flag (FIN probing)

- **FIN Probing:** This flag indicates the end of communication and a single FIN packet is never expected without a connection being previously established. An attacker sends a FIN packet to a port for closing a connection and waits for a response. If the port is open on the target system, the FIN packets will be ignored. If the port is closed, an RST packet will be sent back to the attacker. For example, Windows OS responds to a FIN packet with an RST packet, this clearly indicates the machine is running Windows OS. The FIN probing has the ability to pass undetected through most firewalls, packet filters, and scan detection programs. Here the two cases of FIN flags (normal and abnormal) are illustrated, Fig. 8 shows the normal pattern of FIN-ACK flags and Fig. 9 shows the FIN probing pattern which can be used for the purpose of the fingerprinting attack.
- **FIN/SYN Probing:** These two flags SYN and FIN are not normally set in the same TCP segment header because their purposes are mutually exclusive. An attacker sends a FIN/SYN packet to a port and waits for a response. For example, Linux OS responds to a FIN/SYN packet with a FIN/SYN/ACK packet. This flag combination can be used for the purpose of the fingerprinting attack.
- **URG/PSH/FIN Probing:** This is also known as a Xmas scan which uses a loophole with the TCP RFC 793 to differentiate between open and closed ports. An attacker sends a URG/PSH/FIN packet to a port and waits for a response. If the port is open on the target system then the URG/PSH/FIN packets will be ignored. If the port is closed then an RST/ACK packet will be sent back to the attacker. This Xmas probing does not work with various operating systems that do

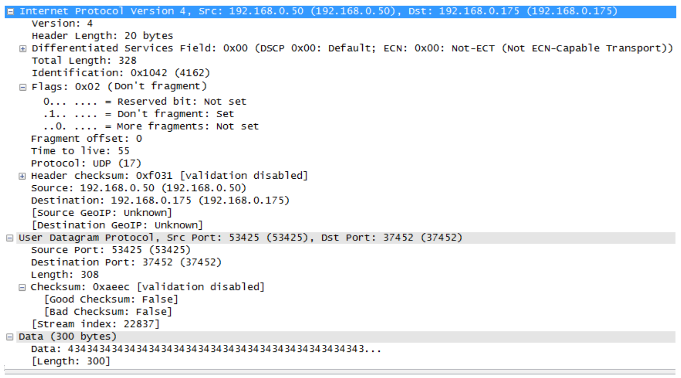
not conform to RFC 793. This flag combination can be used for the purpose of the fingerprinting attack.

- **NULL Packet:** A NULL probing is similar to XMAS probing and FIN probing in its limitations and response. The TCP NULL scan uses a series of uniquely configured TCP packets that contain a sequence number but no flags. If the port is open on the target system then the NULL packets will be ignored. If the port is closed then an RST packet will be sent to the attacker. This flag can be used for the purpose of the fingerprinting attack.
- **Reserved Bit Probing:** The TCP header has 3 reserved bits for future use and these bits should be set to zero. These bits can be used for the purpose of the fingerprinting attack.
- **ECN-Echo Probing:** Explicit Congestion Notification (ECN) is an extension to the TCP packet that allows end-to-end notification of network congestion without dropping packets. ECN is an optional feature that may be used between two ECN-enabled endpoints. This flag can be used for the purpose of the fingerprinting attack.

B. Detection Mechanism based on the Investigation of UDP and ICMP Packets

Many fingerprinting tools utilise ICMP and UDP packets for performing the fingerprinting attack on the target system/network. This ICMP and UDP based fingerprinting technique is different from TCP based fingerprinting and does not use several flags.

- **UDP Requests:** UDP protocol is a connectionless protocol allowing a client to send packets to a UDP service without first establishing a connection. Therefore, UDP works with ICMP for its data transmission. A UDP packet is sent to a port of the target host to check the status of the port. If the UDP port is open on the target system, the packet is accepted without sending a response. If the UDP port is closed, an ICMP error message such as Destination Unreachable is returned to the attacker. When a UDP packet with DF bit set is sent to a UDP port of the target host, the packet can cause the target system to generate an ICMP error message. This is the most common usage of UDP packet for the purpose of the fingerprinting attack.
- **ICMP Inbound Requests:** ICMP request packets are used for legitimate reasons such as to check internal or external address availability or routes, therefore, they can also be used by the attacker to probe and collect substantial information about target OS and devices. An attacker can send specially crafted ICMP echo requests such as ICMP Echo Request (Type 8), ICMP Router Solicitation Request (Type 10) and ICMP Timestamp Request (Types 13) for the fingerprinting attack. These ICMP requests differ from normal ICMP echo requests in that they contain no payload. Usually an ICMP echo request will contain a timestamp, a sequence number and a series of alphabetic characters.



```
⊟ Internet Protocol Version 4, Src: 192.168.0.50 (192.168.0.50), Dst: 192.168.0.175 (192.168.0.175)
    Version: 4
    Header Length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 328
    Identification: 0x1042 (4162)
  ⊟ Flags: 0x02 (Don't fragment)
      0... .... = Reserved bit: Not set
      .1.. .... = Don't fragment: Set
      ..0. .... = More fragments: Not set
    Fragment offset: 0
    Time to live: 55
    Protocol: UDP (17)
  ⊞ Header checksum: 0xf031 [validation disabled]
    Source: 192.168.0.50 (192.168.0.50)
    Destination: 192.168.0.175 (192.168.0.175)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
⊟ User Datagram Protocol, Src Port: 53425 (53425), Dst Port: 37452 (37452)
    Source Port: 53425 (53425)
    Destination Port: 37452 (37452)
    Length: 308
  ⊟ Checksum: 0xaeec [validation disabled]
      [Good Checksum: False]
      [Bad Checksum: False]
    [Stream index: 22837]
⊟ Data (300 bytes)
    Data: 434343434343434343434343434343434343434343434343...
    [Length: 300]
```

Fig. 10. Wireshark capturing UDP packets with Don't Fragment (DF) Bit Set

However, these inbound ICMP request *Types* could also be used against the attacker to locate the potential fingerprinting attacks.

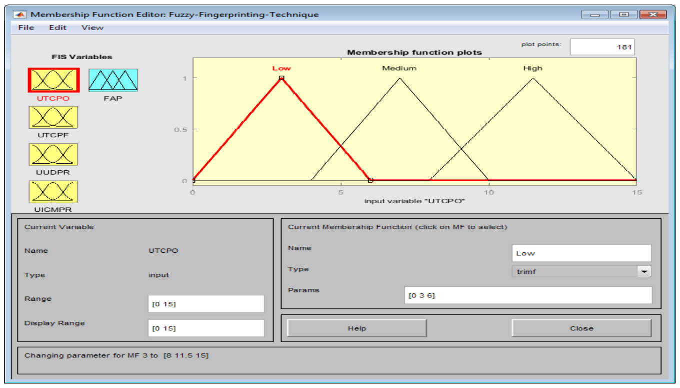
V. Proposed Fuzzy Technique for Detecting the Fingerprinting Attack on the Low-Interaction Honeypot

The real problem with all low-interaction honeypots is that they may not be able to record all the fingerprinting attempts if the attacker uses sophisticated attack techniques or if the honeypot itself is not capable or configured for that attempt. While a honeypot can capture all the fingerprinting attempts, it is extremely difficult to correlate them to conclude that all these attempts are made only for the fingerprinting attack and not for other malicious intentions. Therefore, low-interaction honeypots require an intelligent mechanism to detect fingerprinting attempts and predict the likelihood of fingerprinting of the honeypot. In common with other low-interaction honeypots, KFSensor is susceptible to the fingerprinting attack and sending its identity information to the attacker. Therefore this proposed fuzzy technique offers an intelligent solution based on the detection mechanism proposed in the previous section.

A. Fuzzy Input Variables

The previous section investigated TCP, UDP and ICMP packets and proposed a detection mechanism, which led to the development of the proposed fuzzy technique to detect the signs of the fingerprinting attack and establish its severity level. Based on the proposed detection mechanism parameters for the fingerprinting attack, four fuzzy input variables are derived to observe the four unusual behaviours of TCP, UDP and ICMP packets. These fuzzy input variables are: Unusual TCP Options (UTCPO), Unusual TCP Flags (UTCPF), Unusual UDP Requests (UUDPR) and Unusual ICMP Requests (UICMPR).

Based on the detailed investigation and experimentation in the previous sections, these four fuzzy input variables UTCPO, UTCPF, UUDPR and UICMPR have been assigned the value range 1-15 packets based on the underlying principles of fingerprinting tools Nmap and Xprobe2. Furthermore, these four variables UTCPO, UTCPF, UUDPR and UICMPR are divided into three fuzzy sets: Low, Medium and High, fingerprinting attack categories where the value range of Low is 0-6 packets, the value range of Medium is 4-10 packets, and the value



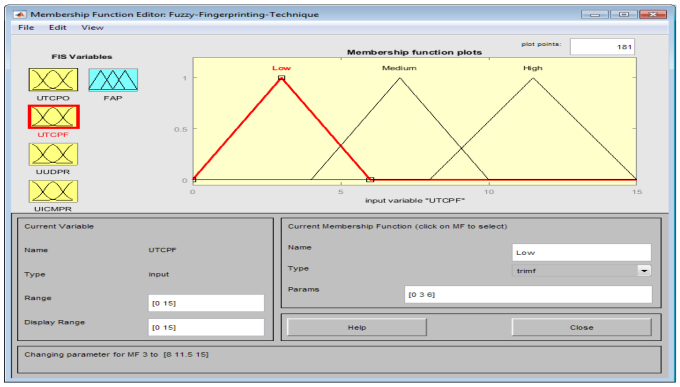
plot points: 181

Current Variable: Name UTCPO; Type input; Range [0 15]; Display Range [0 15]

Current Membership Function (click on MF to select): Name Low; Type trimf; Params [0 3 6]

Changing parameter for MF 3 to [8 11.5 15]

Fig. 11. Fuzzy input variable UTCPO and its fuzzy sets



plot points: 181

Current Variable: Name UTCPF; Type input; Range [0 15]; Display Range [0 15]

Current Membership Function (click on MF to select): Name Low; Type trimf; Params [0 3 6]

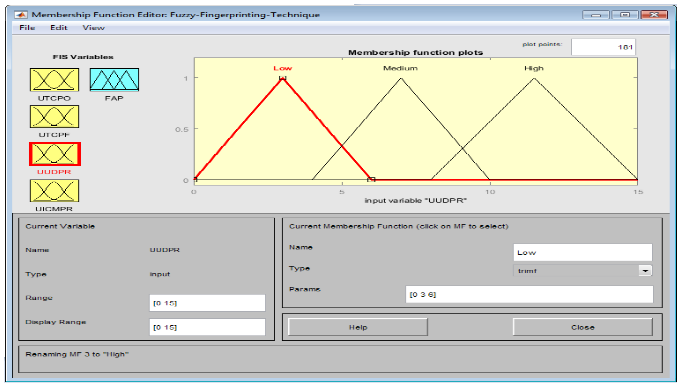
Changing parameter for MF 3 to [8 11.5 15]

Fig. 12. Fuzzy input variable UTCPF and its fuzzy sets

range of High is 8-15 packets. The Matlab design of these four fuzzy input variables UTCPO, UTCPF, UUDPR and UICMPR are shown in Figs. 11 to 14, where fuzzy sets are selected as a triangular membership function for all four fuzzy input variables.

B. Fuzzy Output Variable

These four fuzzy input variables UTCPO, UTCPF, UUDPR and UICMPR are used to generate the output from the fuzzy reasoning system which is the probability of the fingerprinting attack. This fuzzy output variable is called Fingerprinting Attack Probability (FAP). The FAP is represented as a percentage



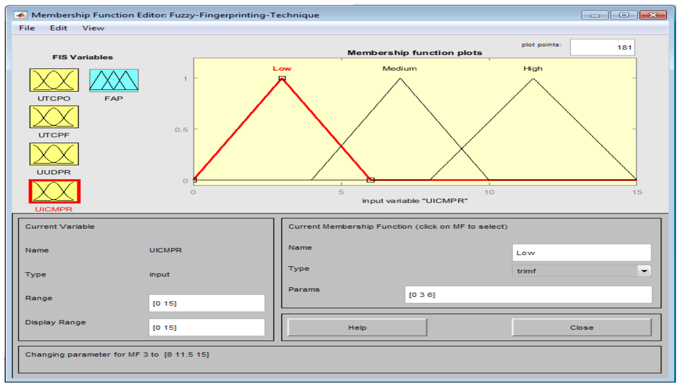
plot points: 181

Current Variable: Name UUDPR; Type input; Range [0 15]; Display Range [0 15]

Current Membership Function (click on MF to select): Name Low; Type trimf; Params [0 3 6]

Renaming MF 3 to "High"

Fig. 13. Fuzzy input variable UUDPR and its fuzzy sets



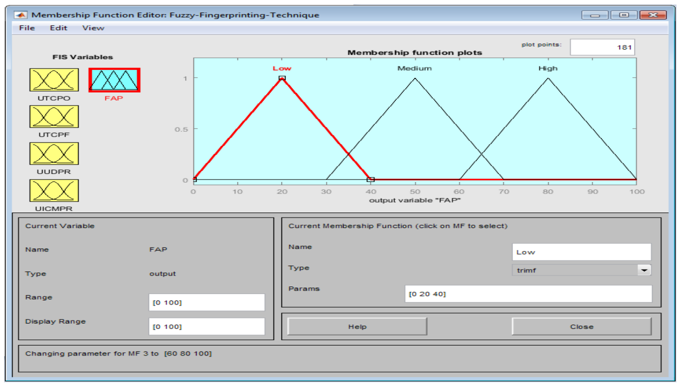
plot points: 181

Current Variable: Name UICMPR; Type input; Range [0 15]; Display Range [0 15]

Current Membership Function (click on MF to select): Name Low; Type trimf; Params [0 3 6]

Changing parameter for MF 3 to [8 11.5 15]

Fig. 14. Fuzzy input variable UICMPR and its fuzzy sets



plot points: 181

Current Variable: Name FAP; Type output; Range [0 100]; Display Range [0 100]

Current Membership Function (click on MF to select): Name Low; Type trimf; Params [0 20 40]

Changing parameter for MF 3 to [60 80 100]

Fig. 15. Fuzzy output variable FAP and its fuzzy sets

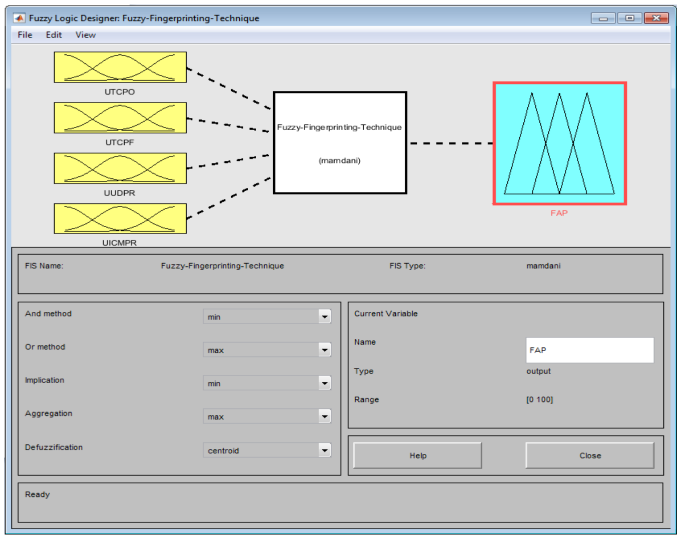
and its entire range (0-100%) is also divided into three fuzzy sets: Low, Medium and High, fingerprinting attack categories with their corresponding ranges 0-40%, 30-70% and 60-100% respectively. The Matlab design of this fuzzy output variable FAP is shown in Fig. 15, where fuzzy sets are also chosen as a triangular membership function for this fuzzy output variable.

C. Fuzzy Rule Base and Fuzzy Reasoning System

The four input and one output variable described above and their corresponding value ranges are utilised in the development of a fuzzy reasoning system (based on Mamdani's inference [23]) as shown in Fig. 16. The fuzzy rules are educed based on the four chosen fuzzy input variables, a sample of fuzzy rules is shown in Fig. 17. Finally, the fuzzy rule base is obtained for the reasoning purposes, a sample of fuzzy rule base is shown in Fig. 18. The development of this fuzzy rule base and fuzzy reasoning system predicts the probability of the fingerprinting attack on the low-interaction honeypot. The complete working procedure of this fuzzy technique is shown in Fig. 19.

VI. Test Results of the Proposed Fuzzy Technique for Predicting the Fingerprinting Attack on the Low-Interaction Honeypot

This section presents the test results of the proposed fuzzy technique for the simulated fingerprinting attacks on the low-interaction honeypot under various controlled network conditions. Table I shows the results of the fingerprinting attack on


Fuzzy Logic Designer: Fuzzy-Fingerprinting-Technique
File
Edit
View
UTCPO
UTCPF
UUDPR
UICMPR
Fuzzy-Fingerprinting-Technique
(mamdani)
FAP
Help
Close

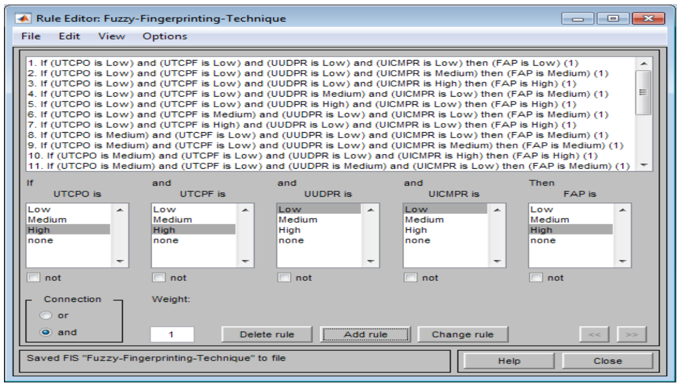
FIS Name: Fuzzy-Fingerprinting-Technique FIS Type: mamdani

And method: min
Or method: max
Implication: min
Aggregation: max
Defuzzification: centroid

Current Variable
Name: FAP
Type: output
Range: [0 100]

Ready

Fig. 16. Fuzzy reasoning system consisting of input and output variables for the proposed fuzzy technique



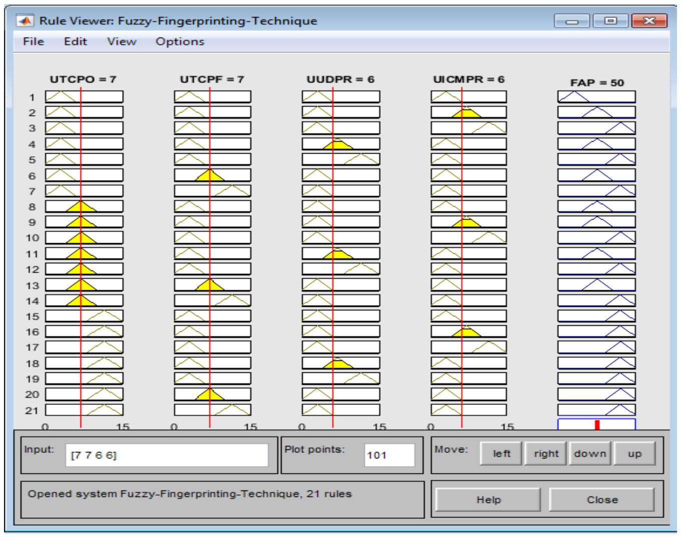
```
1. If (UTCPO is Low) and (UTCPF is Low) and (UUDPR is Low) and (UICMPR is Low) then (FAP is Low) (1)
2. If (UTCPO is Low) and (UTCPF is Low) and (UUDPR is Low) and (UICMPR is Medium) then (FAP is Medium) (1)
3. If (UTCPO is Low) and (UTCPF is Low) and (UUDPR is Low) and (UICMPR is High) then (FAP is High) (1)
4. If (UTCPO is Low) and (UTCPF is Low) and (UUDPR is Medium) and (UICMPR is Low) then (FAP is Medium) (1)
5. If (UTCPO is Low) and (UTCPF is Low) and (UUDPR is High) and (UICMPR is Low) then (FAP is High) (1)
6. If (UTCPO is Low) and (UTCPF is Medium) and (UUDPR is Low) and (UICMPR is Low) then (FAP is Medium) (1)
7. If (UTCPO is Low) and (UTCPF is High) and (UUDPR is Low) and (UICMPR is Low) then (FAP is High) (1)
8. If (UTCPO is Medium) and (UTCPF is Low) and (UUDPR is Low) and (UICMPR is Low) then (FAP is Medium) (1)
9. If (UTCPO is Medium) and (UTCPF is Low) and (UUDPR is Low) and (UICMPR is Medium) then (FAP is Medium) (1)
10. If (UTCPO is Medium) and (UTCPF is Low) and (UUDPR is Low) and (UICMPR is High) then (FAP is High) (1)
11. If (UTCPO is Medium) and (UTCPF is Low) and (UUDPR is Medium) and (UICMPR is Low) then (FAP is Medium) (1)
```

If UTCPO is	and UTCPF is	and UUDPR is	and UICMPR is	Then FAP is
Low	Low	[✓ Low]	[✓ Low]	Low
Medium	Medium	Medium	Medium	Medium
[✓ High]	[✓ High]	High	High	[✓ High]
none	none	none	none	none
not	not	not	not	not

Connection: or [✓ and]
Weight: 1

Saved FIS "Fuzzy-Fingerprinting-Technique" to file

Fig. 17. Fuzzy rules of the proposed fuzzy technique to predict the fingerprinting attack on the low-interaction honeypot



Input: [7 7 6 6]
Plot points: 101
Move:

Opened system Fuzzy-Fingerprinting-Technique, 21 rules

Fig. 18. Fuzzy rule base of the proposed fuzzy technique to predict the fingerprinting attack on the low-interaction honeypot

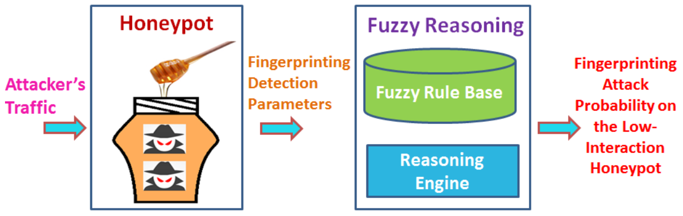

Attacker's Traffic
Honeypot
Fingerprinting Detection Parameters
Fuzzy Reasoning
Fuzzy Rule Base
Reasoning Engine
Fingerprinting Attack Probability on the Low-Interaction Honeypot

Fig. 19. Illustration of the working procedure for the proposed fuzzy technique to predict the fingerprinting attack on the low-interaction honeypot

KFSensor honeypot for all the previous five fingerprinting attack conditions explained in the Section III. These five attacks are performed on the KFSensor using the Nmap fingerprinting tool which is extensively covered in the previous analysis while developing this approach, thus, the method encompasses the underlying attack principles of Nmap. This is evident from the test results given in the Table I.

A total 50 fingerprinting attacks are carried out and 10 repetition for every fingerprinting attack. Out of 50 attacks, the fuzzy technique predicted 47 attacks as high or medium attack probability, where 31 attacks are predicted as high and 16 are predicted as medium. Only 3 attacks are predicted as low probability attacks by the fuzzy technique. For the four special OS fingerprinting attack scripts 1,2,3 and 5 in Table I, out of 40 attacks, 31 attacks are predicted as high probability attacks and 9 attacks are medium attacks with the accuracy of 77.5%. The fourth fingerprinting attack script (*nmap -sV –version-intensity 9 192.168.0.175*) is different from the other four scripts, by utilising a different nmap database *nmap-services* [24] while the other four scripts use *nmap-os-db* [25]. It detects the operating system and device type but based on the detected service information such as identify a service as Microsoft Exchange means the operating system is Windows because Microsoft Exchange runs on Windows OS [21]. Despite the fourth fingerprinting attack script being a different type of indirect OS fingerprinting attack script, the fuzzy technique predicted 7 times medium probability of attack in 10 attacks.

This preliminary test results are encouraging and indicate that the fuzzy technique can identify the majority of the Nmap fingerprinting attacks successfully. However, it requires the inclusion of more parameters related to application services to encompass all fingerprinting attacks. Moreover, the controlled network conditions reveal that the success of the fuzzy technique also depends on some external factors such as all the fingerprinting probe packets being received successfully and timely, otherwise, it would affect the analysis and prediction result of the fuzzy technique. This is reflected in the results of the scripts 1,3 and 5 where the 3 prediction results of each script are medium but not high. Therefore, it should be noted that the same type of fingerprinting test may yield a different result depending on the probe packet sent to the target and received by the target in the configured network.

VII. Conclusion

This paper presented a fuzzy technique for detecting and predicting the fingerprinting attack on the low-interaction honeypot. Initially, it presented an experimental assessment of the fingerprinting attack on the low-interaction honeypot

TABLE I. Proposed Fuzzy Technique based Detection and Prediction of the fingerprinting attack on the KFSensor Honeypot

No.	Nmap Fingerprinting Attack Script	Fingerprinting Attack Probability for 10 Run		
		Low	Medium	High
1	$nmap\ -O$ 192.168.0.175	0	3	7
2	$nmap\ -A$ 192.168.0.175	0	0	10
3	$nmap\ -O\ -fuzzy\ --osscan-guess$ 192.168.0.175	0	3	7
4	$nmap\ -sV\ --version-intensity$ 9 192.168.0.175	3	7	0
5	$nmap\ -O\ --max-OS-tries$ 5 192.168.0.175	0	3	7

called KFSensor. Subsequently, based on this experimental assessment, the paper proposed a fingerprinting detection mechanism for the low-interaction honeypot, which includes the underlying principles of popular fingerprinting attack tools. Finally, based on the proposed fingerprinting detection mechanism, the paper proposed a fuzzy technique for detecting and identifying the fingerprinting attack on the low-interaction honeypot. This proposed fuzzy technique is a general technique and can be used with any low-interaction honeypot to aid in the identification of the fingerprinting attack whilst it is occurring; thus it can protect the honeypot from the fingerprinting attack and extend its life. Despite the fuzzy technique being promising, it may not be effective in cases where some unknown fingerprinting probes are used which are not covered in the proposed detection method. Therefore, in the future, it is important to extend this technique and include some additional attack principles. Furthermore, it is beneficial to transform this fuzzy technique as an adaptive technique based on the Dynamic Fuzzy Rule Interpolation (D-FRI) framework [26], [27], [28], [29], [30] and adaptive FRI [31], [32], [33], [34], thus, it can be more effective for changing attack patterns.

References

[1] R. Joshi and A. Sardana, *Honeypots: A new paradigm to information security*. CRC Press, 2011.

[2] R. A. Grimes, *Honeypots, in Hacking the Hacker: Learn from the Experts Who Take Down Hackers*. John Wiley & Sons, Inc., Indianapolis, Indiana, 2017. [Online]. Available: doi:10.1002/9781119396260.ch19

[3] L. Spitzner, *Honeypots: Tracking Hackers*. Addison-Wesley Reading, 2003, vol. 1.

[4] J. M. Allen. (2008) OS and Application Fingerprinting Techniques. [Online]. Available: https://www.sans.org/reading-room/whitepapers/authentication/os-application-fingerprinting-techniques-32923

[5] N. Naik and P. Jenkins, "Fuzzy reasoning based windows firewall for preventing denial of service attack," in *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2016, pp. 759–766.

[6] ——, "Enhancing windows firewall security using fuzzy reasoning," in *IEEE International Conference on Dependable, Autonomic and Secure Computing*, 2016, pp. 263–269.

[7] N. Naik, P. Jenkins, R. Cooke, D. Ball, A. Foster, and Y. Jin, "Augmented windows fuzzy firewall for preventing denial of service attack," in *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE, 2017, pp. 1–6.

[8] N. Naik, R. Diao, C. Shang, Q. Shen, and P. Jenkins, "D-FRI-WinFirewall: Dynamic fuzzy rule interpolation for windows firewall," in *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE, 2017, pp. 1–6.

[9] N. Naik, "Fuzzy inference based intrusion detection system: FI-Snort," in *IEEE International Conference on Dependable, Autonomic and Secure Computing*, 2015, pp. 2062–2067.

[10] N. Naik, R. Diao, and Q. Shen, "Application of dynamic fuzzy rule interpolation for intrusion detection: D-FRI-Snort," in *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2016, pp. 78–85.

[11] ——, "Dynamic fuzzy rule interpolation and its application to intrusion detection," *IEEE Transactions on Fuzzy Systems*, 2017.

[12] L. Yang, J. Li, G. Fehringer, P. Barraclough, G. Sexton, and Y. Cao, "Intrusion detection system by fuzzy interpolation," in *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2017, pp. 1–6.

[13] N. Naik, P. Jenkins, N. Savage, and V. Katos, "Big data security analysis approach using computational intelligence techniques in R for desktop users," in *IEEE Symposium Series on Computational Intelligence (SSCI)*, 2016.

[14] N. Naik and P. Jenkins, "A fuzzy approach for detecting and defending against spoofing attacks on low interaction honeypots," in *2018 21st International Conference on Information Fusion (Fusion)*, 2018.

[15] N. C. Rowe, "Measuring the effectiveness of honeypot counter-counterdeception," in *System Sciences, 2006. HICSS'06. Proceedings of the 39th Annual Hawaii International Conference on*, vol. 6. IEEE, 2006, pp. 129c–129c.

[16] G. F. Lyon. (2009) Chapter 8. Remote OS Detection: TCP/IP Fingerprinting Methods supported by Nmap. [Online]. Available: https://nmap.org/book/osdetect-methods.html

[17] L. G. Greenwald and T. J. Thomas, "Toward undetected operating system fingerprinting." *WOOT*, vol. 7, pp. 1–10, 2007.

[18] G. F. Lyon, *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure, 2009.

[19] ——. (2009) Chapter 8. Remote OS Detection: Usage and Examples. [Online]. Available: https://nmap.org/book/osdetect-usage.html

[20] ——. (2009) Chapter 15. Nmap Reference Guide. [Online]. Available: https://nmap.org/book/man-os-detection.html

[21] ——. (2009) Chapter 7. Service and Application Version Detection. [Online]. Available: https://nmap.org/book/vscan.html

[22] ——. (2009) Chapter 15. Service and Version Detection. [Online]. Available: https://nmap.org/book/man-version-detection.html

[23] E. H. Mamdani and S. Assilina, "An experiment in linguistic synthesis with a fuzzy logic controller," *International Journal of Man-Machine Studies*, vol. 7, no. 1, pp. 1–13, 1975.

[24] G. F. Lyon. (2011) Nmap Service DB. [Online]. Available: https://svn.nmap.org/nmap/nmap-services

[25] ——. (2017) Nmap OS Fingerprinting 2nd Generation DB. [Online]. Available: https://svn.nmap.org/nmap/nmap-os-db

[26] N. Naik, P. Su, and Q. Shen, "Integration of interpolation and inference," in *UK Workshop on Computational Intelligence*, 2012, pp. 1–7.

[27] N. Naik, "Dynamic Fuzzy Rule Interpolation," Ph.D. dissertation, Department of Computer Science, Institute of Mathematics, Physics and Computer Science, Aberystwyth University, UK, 2015.

[28] N. Naik, R. Diao, C. Quek, and Q. Shen, "Towards dynamic fuzzy rule interpolation," in *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2013, pp. 1–7.

[29] N. Naik, R. Diao, and Q. Shen, "Genetic algorithm-aided dynamic fuzzy rule interpolation," in *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2014, pp. 2198–2205.

[30] ——, "Choice of effective fitness functions for genetic algorithm-aided dynamic fuzzy rule interpolation," in *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2015, pp. 1–8.

[31] L. Yang and Q. Shen, "Adaptive fuzzy interpolation," *IEEE Transactions on Fuzzy Systems*, vol. 19, no. 6, pp. 1107–1126, 2011.

[32] ——, "Closed form fuzzy interpolation," *Fuzzy Sets and Systems*, vol. 225, pp. 1–22, 2013.

[33] L. Yang, F. Chao, and Q. Shen, "Generalized adaptive fuzzy rule interpolation," *IEEE Transactions on Fuzzy Systems*, vol. 25, no. 4, pp. 839–853, 2017.

[34] J. Li, L. Yang, X. Fu, F. Chao, and Y. Qu, "Dynamic QoS solution for enterprise networks using tsk fuzzy interpolation," in *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE, 2017, pp. 1–6.