Hiding Computer Network Proactive Security Tools Unmasking Features

Roman V. Maximov, Sergey P. Sokolovsky, Alexey L. Gavrilov
Shtemenko Krasnodar Higher Military School
Krasnodar, Russia
rvmaxim@yandex.ru; ssp.vrn@mail.ru; aleks02.94@mail.ru

***Abstract*—Passive and proactive network security tools, based on cyber deception technologies, become more and more popular among classic tools. Using such tools gives an opportunity to prevent network attacks on the very beginning – at intelligence gathering stage. In this work we research one of these deceptive tools – a network tarpit. Based on *LaBrea* taprit, we investigate some fingerprints of its algorithms, that may lead to tarpit detecting and lowering overall security level. We used an open source detection tool *Degreaser* to find *LaBrea's* unmasking features, classify them and calculate their influence on the possibility of tarpit discovering. Our main goal was to provide methods to improve network tarpit obscuring capabilities, ridding of revealed unmasking features. These methods were later implemented as modules and integrated in our network tarpit called *NetHole*, that uses *LaBrea* as prototype and has no revealed shortcomings. The efficiency of modifications made was then tested in a set of tests with the same detection tool *Degreaser*.**

Keywords—information security; proactive defence; network tarpit; network security tools; unmasking features; cyber deception; network intelligence gathering

I. INTRODUCTION

Large part of modern network attacks is being conducted for intelligence gathering issues, to reveal the topology of the network being attacked and security tools being used in this network [1-3]. Automated scanning tools are used in such attacks with high possibility.

Among other security tools there is a subclass based on deception tactics [4-7]. Its main idea is not in increasing the power or the amount of tools being used but to provide illusions about network topology, thus slowing automated scanners and confusing manual attackers. One of such applications is so called honeypot [8-21]. More complex ways of network deception include not only topology illusion and false vulnerable hosts production but proactive defense in addition, e.g. trapping connections with attacker, exhausting his resources for maintaining connection state. A large number of such trapped connections lead to slowdowning of automated network scanners or even may cause an impossibility of overall network interaction for the intruder caught by. These tools, called network tarpits [22, 23], may work as standalone deception applications or be included in firewall packets, e.g. *Linux Netfilter Tarpit*, part of *Xtables-addons* [24].

II. COMPROMISING FEATURES

One of the main advantages of deceptive network security tools is their invisibility. Intruders, in their turn, actively create new and modify existing tools for uncovering honeypots and network tarpits, making them useless. Deceptive security tools can be compromised by detecting their unique fingerprints, which can be also called unmasking features. To achieve this, attackers may use either common network traffic analyzers as *nmap*, *zenmap*, *ethereal*, *arping*, *tethreal*, etc, or special tools, developed for discovering proactive security tools [25, 26].

We used one of such special tools, *Degreaser* [25, 26], for testing *LaBrea* and informativity of its fingerprints and figured out the following two types of unmasking features:

1) Unreliable features, that can't be a total evidence of tarpit presence. They are:

a) A hardcoded MAC-address: *LaBrea* uses $(00:00:0F:FF:FF:FF)_{16}$ address regardless of physical address of network adapter it works on. Network tarpits are often used against threats outside *LAN*, where layer-2 address cannot be seen. Moreover, multiply IP-addresses can be assigned for single network interface.

b) Opened TCP ports: LaBrea answers to requests to all *TCP*-ports of fake host, resulting in all *TCP*-ports to seem opened. There are 65536 possible ports on every host and it takes 2^{16} requests per host to check every port on it. Such scanning cost is too high.

c) Delayed response: There is a time delay between ARP-request and response in *LaBrea* promiscuous mode. It's also a secondary fingerprint, because there always can be interferences causing such delays.

2) Discriminating features, which lead to reliable tarpit detection. They are:

d) TCP window size: Fundamental feature of tarpit-like host is manipulation with *TCP*-window size. Default window size used by *LaBrea* is 10 bytes. This value is configurable, but only once before running the tarpit. Small size of *TCP* window is the first sign of tarpit presence.

e) TCP options: TCP options can be used by hosts to negotiate additional functionality. Typically, these options are set by operating systems during establishing *TCP* connection. *LaBrea* establishes its own *TCP* sessions, bypassing system level, so it has to manage *TCP* options itself, but it is not

implemented. Ignoring *TCP* options is a second significant fingerprint of *LaBrea* presence.

So, there should be a balance between effectiveness of network protection tools and possibility of them to be discovered using their unmasking features. The main goal of our work is to find that balance, developing methods to decrease the level of informativity of network tarpits' unmasking features.

III. NETHOLE

We investigated *Degreaser* source code to find out the fingerprints it searches for and created *NetHole*, that has no unmasking features listed above, using *LaBrea* as prototype.

The first method to lower the possibility of uncovering network tarpit being used is in the following.

A. Address space

The set of available *IP* addresses is divided preliminarly into subsets of authorized and used (connected) addresses of network devices, authorized and temporarily unused network addresses, the rest of set is marked as forbidden to be used by network tarpit (Table I). The main idea of this method is to increase the functioning realism of protected network.

Dividing all *IP* addresses in such set will not cause a situation of a network, where every address is available like a false host, thus lowering the possibility of used network tarpit being revealed. In addition, all attempts to establish connection with hosts with *IP* addresses from forbidden set can be identified as network topology scanning or attacks themselves.

TABLE I. DIVIDING ADDRESS SPACE

<table>
<tr><td colspan="3">The set of all available IP-addresses</td></tr>
<tr><td colspan="2">Authorized IP addresses subset</td><td rowspan="2">Forbidden
IP addresses subset</td></tr>
<tr><td>Used IP addresses subset (real network devices)</td><td>Temporarily unused IP addresses subset</td></tr>
</table>

B. Randomizing MAC address

A hardcoded *MAC* address used by *LaBrea* (fig. 1) is a clear network tarpit fingerprint. *MAC* address is a unique 6-bytes number used for identification of Ethernet frames sender and receiver that is set by the manufacturer of network adapter.

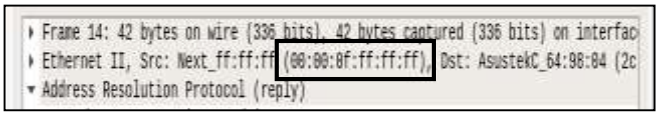

Fig. 1. The only hardcoded MAC address used by LaBrea

To hide this fingerprint, we used random *MAC* addresses for every fake host. We suggest 3 different options for a physical addresses generator:

1) Fully random MAC address: First option supposes generating completely random *MAC* address for every fake host. To start with, we make an array of physical addresses of currently active local network devices. Then, we generate random sequence of hexadecimal numbers of size *J*, where *J* is an amount of false hosts needed. To avoid the situation with duplicate *MAC* adresses in one network segment, we need to check whether every generated address is already in use by any of real network devices. This address must not also be null $(00:00:00:00:00:00)_{16}$ or broadcast $(ff:ff:ff:ff:ff:ff)_{16}$. While at least one of these conditions is true, address will be regenerated and then checked again. After that we align every *i*-th *MAC* address with a *j*-th *IP* address of false host.

The general algorithm of network tarpit with this modification is described next. When *ARP* request to any *i*-th *IP* address from given set is received, and if this *IP* address is from temporarily unused addresses subset, response packet is generated with *TCP* window size of 10 bytes and the aligned *j*-th *MAC* address in *TCP* header. This packet is send then to initial sender on behalf of fake host.

The described method was implemented in NetHole tarpit, its effectiveness was tested in a series of tests, the main purpose of which was to compare the discovering rate of LaBrea and NetHole. To identify the network tarpit, i.e. the unmasking feature of used security tool, we analyzed the intercepted ARP packets with Wireshark. Table (II) represents the dump of ARP protocol. The response on request "What MAC address does the host with i-th IP address have?" is "The i-th IP address is set to the host with j-th MAC address", where j-th MAC address is randomly generated.

Partially random MAC-address: The second option of described method suggests using a database with unique vendors MAC octets (table III). The three upper octets of generating MAC address are got from this database, the rest remain random, as described in the first option. In order the imitating network to seem more realistic, the vendor is chosen randomly every time. Using this option leads to resulting false network to contain devices made by real companies.

TABLE II. USING OF GENERATED MAC ADDRESSES

Sender	**Receiver**	**Contents of ARP-request**
Router	Broadcast	Who has 10.0.0.40? Tell 10.0.0.100
be:97:a6:1c:2a:ef	Router	10.0.0.40 is at be:97:a6:1c:2a:ef
Router	Broadcast	Who has 10.0.0.41? Tell 10.0.0.100
d4:e2:da:95:eb:3f	Router	10.0.0.41 is at d4:e2:da:95:eb:3f
Router	Broadcast	Who has 10.0.0.42? Tell 10.0.0.100
3e:8b:0b:d0:dd:ae	Router	10.0.0.42 is at 3e:8b:0b:d0:dd:ae

TABLE III. THE PART OF UNIQUE VENDORS MAC OCTETS DATABASE

MAC-address	Vendor
000142	Cisco Systems, Inc
00037F	Atheros Communications, Inc
000393	Apple, Inc.
0004EA	Hewlett Packard
0004E9	Infiniswitch Corporation
000585	Juniper
00059E	Zinwell Corporation

2) Partially random MAC-address with real percentage ratio: The third option assumes preliminary ARP-scanning of protected network in order to identify vendors of the local devices by their MAC-addresses using the database described above. It gives us a percentage ratio of used network devices, using which we can imitate the most true-to-life false network.

C. Implementing TCP-options

The *LaBrea* never adds *TCP* options in headers of generated response packets. To improve believability of answers being sent, we decided to add the *TCP* option support. This feature reads all options from incoming request packet and copies them into the response packet, excluding "*TCP Timestamp*" option. This option contains two 4-byte fields with timestamps. The «*Timestamp Value*» (*Tsval*) field contains the packet sender's current value of timestamp. Firstly, it's copied to «*Timestamp Echo Reply*» (*Tsecr*) field and then the current system uptime value is written in it.

We used *Degreaser* to test this feature. While scanning, *Degreaser*, among other tests, checks presence of any *TCP* options in response packets. Table (IV) contains the output of network scanning with *LaBrea* running in it.

Degreaser identifies scanned hosts as network tarpits. In the "*TCP options*" column found options are given. As stated at table (IV), all hosts which are considered to be the "*LaBrea tarpit*" have no *TCP*-options. Table (V) contains the results of network testing with *NetHole* working in it The "*TCP options*" column now contains *Maximum Segment Size (M), Windows Scale (W), Selective Acknowledgement (S) and Timestamp (T)* options. *Degreaser* cannot discover that all these hosts are held by network tarpit.

D. Random TCP-window size

Network tarpits use *TCP* flow control to catch attackers by changing the *TCP* window size, so it can be attributed to other uncovering features. *LaBrea* sets *TCP* window size to 10 bytes by default. The *Degreaser's* algorithm checks this parameter after checking *TCP* options, and if it is less than control value, the host is considered to be a potential tarpit.

TABLE IV. THE NETWORK WITH LABREA SCANNING RESULTS

IP Address	Response Time	Window Size	TCP Flags	TCP Options	Scan Result
212.193.1.210	0	0	–	–	No response
212.193.1.92	257762	10	SA	–	LaBrea
212.193.1.198	0	0	–	–	No response
212.193.1.105	81284	10	SA	–	LaBrea
212.193.1.195	205014	10	SA	–	LaBrea
212.193.1.121	0	0	–	–	No response
212.193.1.251	0	0	–	–	LaBrea
212.193.1.226	127510	10	SA	–	LaBrea

In *NetHole* we use a small (up to 255 bytes) but random *TCP* window size. It may be a result of additional traffic received by tarpit, but it hides the unmasking fingerprint at the same time. On the fragment of *TCP* dump below the randomly generated window size of 195 bytes is highlighted.

```
Source Port: 22
Destination Port: 48414
[Stream index: 5]
[TCP Segment Len: 0]
Sequence number: 0 (relative sequence number)
Acknowledgment number: 1 (relative ack number)
Header Length: 20 bytes
Flags: 0x012 (SYN, ACK)
Window size value: 195
[Calculated window size: 195]
[SEQ/ACK analysis]
```

TABLE V. THE NETWORK WITH NETHOLE SCANNING RESULTS

IP Address	Response Time	Window Size	TCP Flags	TCP Options	Scan Result
212.193.1.49	136629	10	SA	MVST	Real host
212.193.1.70	0	0	–	–	No response
212.193.1.144	99569	10	SA	MVST	Real host
212.193.1.125	0	0	–	–	No response
212.193.1.196	0	0	–	–	No response
212.193.1.233	0	0	–	–	No response
212.193.1.88	227956	10	SA	MVST	Real host
212.193.1.140	0	0	–	–	No response

Degreaser scanning output is shown in table VI.

TABLE VI. SCANNING RESULTS WITH RANDOMLY GENERATED TCP WINDOW SIZE

IP Address	Response Time	Window Size	TCP Flags	TCP Options	Scan Result
212.193.1.144	174390	178	SA	–	Real host
212.193.1.195	149446	230	SA	–	Real host
212.193.1.122	107480	201	SA	–	Real host
212.193.1.141	128513	52	SA	–	Real host
212.193.1.226	0	0	–	–	No response
212.193.1.146	127476	126	SA	–	Real host
212.193.1.230	2295501	196	SA	–	Real host
212.193.1.73	0	0	–	–	No response

E. Application level responses

The main purpose of network tarpits is to hang the network session with attacker as long as possible. *LaBrea* ignores all data packets after *TCP* session is established, compromising itself. *Degreaser* exploits this feature, sending a *TCP* packet with random data, which size is "*TCP window size – 1*", and waits for response. If there is no response, currently scanned host is considered to be a tarpit. In order to hide this feature, we implemented a module for sending confirmation tickets after receiving any packet with data. The idea is to send a *TCP-ACK* packet with adjusted window size in response to *TCP* packet with *PUSH* flag.

The example of network interaction between *Degreaser* (with IP address 212.193.1.10) and *NetHole* (with IP address 212.193.1.28) could be seen in Table (VII). There are five *TCP* packets, three of them were used for connection establishing, the 4th is a 9-bytes data packet and the 5th is a *TCP-ACK* packet sent as a confirmation ticket.

TABLE VII. IMITATION OF APPLICATION LEVEL RESPONSE

Source	Destination	Protocol	Contents
212.193.1.10	212.193.1.10	TCP	32622 – 80 [SYN] Seq=0 Win=5840 Len=0
212.193.1.28	212.193.1.10	TCP	80 - 32622 [SYN, ACK] Seq=0 Ack=1 Win=10 Len=0
212.193.1.10	212.193.1.10	TCP	32622 – 80 [ACK] Seq=1 Ack=1 Win=5840 Len=0
212.193.1.10	212.193.1.10	TCP	32622 – 80 [ACK] Seq=1 Ack=1 Win=5840 Len=9
212.193.1.28	212.193.1.10	TCP	80 - 32622 [ACK] Seq=1 Ack=1 Win=30 Len=0

TABLE VIII. SCANNING RESULTS WITH APPLICATION LEVEL RESPONSES IMPLEMENTED

IP Address	Response Time	Window Size	TCP Flags	TCP Options	Scan Result
212.193.1.193	103529	10	SA	–	Real host
212.193.1.73	0	0	–	–	No response
212.193.1.47	0	0	–	–	No response
212.193.1.67	0	0	–	–	No response
212.193.1.87	207563	10	SA	–	Real host
212.193.1.106	199462	10	SA	–	Real host
212.193.1.105	0	0	–	–	No response
212.193.1.172	0	0	–	–	No response

As listed in table (VIII), there are no *TCP* options in all responses, *TCP* window size is 10 bytes, that is less than default minimal value for *Degreaser*, but all these hosts are not considered to be real hosts, neither common network tarpit, nor *LaBrea* tarpit especially, because of send *TCP-ACK* confirmation packets.

IV. CONCLUSION

Using the tests described above we confirmed that using the developed modules leads to increasing the effectiveness of network tarpit and its stealthiness level through decreasing the possibility of its uncovering and identification by intruders.

REFERENCES

[1] Hayatle O., Youssef A., Otrok H. Dempster-Shafer Evidence Combining for Anti-Honeypot Technologies. Inf. Sec. J.: A Global Perspective 21, 6 (January 2012), 2012, pp. 306-316. DOI: 10.1080/19393555.2012.738375.

[2] Laurén S., Leppänen V., Rauti S., Uitto J. A Survey on Anti-honeypot and Anti-introspection Methods. Recent Advances in Information Systems and Technologies - Volume 2, WorldCIST'17, Porto Santo Island, Madeira, Portugal, April 11-13, 2017, pp. 125-134. DOI: 10.1007/978-3-319-56538-5_13.

[3] Markov A.S., Tsirlov V.L. Guidelines for Cybersecurity in the Context of ISO 27032, Voprosy kiberbezopasnosti *[Cybersecurity issues]*, 2014, No 1 (2). P. 28-35. DOI: 10.21681/2311-3456-2014-1-28-35.

[4] Achleitner S., La Porta T., McDaniel P., Sugrim S., Krishnamurthy S.V., Chadha R. Cyber Deception: Virtual Networks to Defend Insider Reconnaissance. In Proceedings of the 8th ACM CCS International Workshop on Managing Insider Security Threats (MIST '16). ACM, New York, NY, USA, 2016, pp. 57-68. DOI: 10.1145/2995959.2995962.

[5] De Gaspari F., Jajodia S., Mancini L.V., Panico A. AHEAD: A New Architecture for Active Defense. In Proceedings of the 2016 ACM Workshop on Automated Decision Making for Active Cyber Defense (SafeConfig '16). ACM, New York, NY, USA, 2016, pp. 11-16. DOI: 10.1145/2994475.2994481.

[6] Shaw T., Arrowood J., Kvasnicka M., Taylor S., Cook K., Hale J. POSTER: Evaluating Reflective Deception as a Malware Mitigation Strategy. In Proceedings of the 2017 ACM SIGSAC Conference on

Computer and Communications Security (CCS '17). ACM, New York, NY, USA, 2017, pp. 2575-2577. DOI: 10.1145/3133956.3138833.

[7] Almeshekah M.H., Spafford E.H. Planning and Integrating Deception into Computer Security Defenses. In Proceedings of the 2014 New Security Paradigms Workshop (NSPW '14). ACM, New York, NY, USA, 2014, pp. 127-138. DOI: 10.1145/2683467.2683482.

[8] Du Z., Fan W., Fernández D., Villagrá V.A. Enabling an Anatomic View to Investigate Honeypot Systems: A Survey. November 2017. IEEE Systems Journal 11/2017, pp (99):1-14. DOI: 10.1109/JSYST.2017.2762161.

[9] Keil, C., Nawrocki, M., Schmidt, T.C., Schönfelder, J., Wählisch, M.: A Survey on Honeypot Software and Data Analysis. arXiv.org, 2016, vol. 10, pp. 63-75.

[10] Sokol P., Míšek J., Husák M. Honeypots and honeynets: issues of privacy. EURASIP Journal on Information Security.2017, 1, Article 57 (December 2017), 9 pages. DOI: 10.1186/s13635-017-0057-4.

[11] Nawrocki M., Wahlisch M., Schmidt T., Keil C., Schonfelder J. A Survey on Honeypot Software and Data Analysis. 2016. CoRR, abs/1608.06249.

[12] Olagunju A.O., Samu F. In Search of Effective Honeypot and Honeynet Systems for Real-Time Intrusion Detection and Prevention. In Proceedings of the 5th Annual Conference on Research in Information Technology (RIIT '16). ACM, New York, NY, USA, 2016, pp. 41-46. DOI: 10.1145/2978178.2978184.

[13] Han W., Zhao Z., Doupé A., Ahn G. HoneyMix: Toward SDN-based Intelligent Honeynet. In Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization (SDN-NFV Security '16). ACM, New York, NY, USA, 2016, pp. 1-6. DOI: 10.1145/2876019.2876022.

[14] Guarnizo J.D., Tambe A., Bhunia S.S., Ochoa M., Tippenhauer N.O., Shabtai A., Elovici Y. SIPHON: Towards Scalable High-Interaction Physical Honeypots. In Proceedings of the 3rd ACM Workshop on Cyber-Physical System Security (CPSS '17). ACM, New York, NY, USA, 2017, pp 57-68. DOI: 10.1145/3055186.3055192.

[15] Tiwari R. Jain A. Improving network security and design using honeypots. In Proceedings of the CUBE International Information Technology Conference (CUBE '12). ACM, New York, NY, USA, 2012, pp. 847-852. DOI: 10.1145/2381716.2381875.

[16] Andrew D., Chi H. An empirical study of botnets on university networks using low-interaction honeypots. In Proceedings of the 51st ACM Southeast Conference (ACMSE '13). ACM, New York, NY, USA, 2013, Article 44, 2 pages. DOI: 10.1145/2498328.2500094.

[17] Pisarčík P., Sokol P. Framework for distributed virtual honeynets. In Proceedings of the 7th International Conference on Security of Information and Networks (SIN '14). ACM, New York, NY, USA, 2014, Pages 324, 6 pages. DOI: 10.1145/2659651.2659685.

[18] Laurén S., Rauti S., Leppänen V. An interface diversified honeypot for malware analysis. In Proccedings of the 10th European Conference on Software Architecture Workshops (ECSAW '16). ACM, New York, NY, USA, 2016, Article 29, 6 pages. DOI: 10.1145/2993412.2993417.

[19] Saud Z., Islam M.H. Towards proactive detection of advanced persistent threat (APT) attacks using honeypots. In Proceedings of the 8th International Conference on Security of Information and Networks (SIN '15). ACM, New York, NY, USA, 2015, pp. 154-157. DOI: 10.1145/2799979.2800042.

[20] Borkar A., Salunke A., Barabde A., Karlekar N. P. Honeypot: a survey of technologies, tools and deployment. In Proceedings of the International Conference & Workshop on Emerging Trends in Technology (ICWET '11). ACM, New York, NY, USA, 2011, pp. 1357-1357. DOI: 10.1145/1980022.1980327.

[21] Shmatova E. The Choice of Strategy for the Spurious Information System on the Basis of the Game Theory Model. Voprosy kiberbezopasnosti *[Cybersecurity issues]*, 2015. No 5 (13). P. 36-40. DOI: 10.21681/2311-3456-2015-5-36-40.

[22] Liston T. LaBrea: «sticky» Honeypot and IDS. [Online]. Available: http://labrea.sourceforge.net/labrea-info.html.

[23] Liston T. «LaBrea». [Online]. Available: http://labrea.sourceforge.net/labrea.1.txt.

[24] Hopkins A. TARPIT-iptables TARPIT target. [Online]. Available: http: //www.netfilter.org/projects/patch-o-matic/pom-external.html.

[25] Alt. L. Degreaser git respository. 2014. https://github.com/lancealt/degreaser.

[26] Alt L., Beverly R., Dainotti A. Uncovering network tarpits with degreaser. In Proceedings of the 30th Annual Computer Security Applications Conference (ACSAC '14). ACM, New York, NY, USA, 2014, pp. 156-165. DOI: 10.1145/2664243.2664285.