MitiBox: Camouflage and Deception for Network Scan Mitigation

Erwan Le Malécot[†‡]
[†]*Kyushu University*
[‡]*Institute of Systems, Information Technologies and Nanotechnologies*

Abstract

Reconnaissance, if successful, provides a definite tactical advantage in a battle and, as such, unsolicited computer network scans are often the precursors to more significant attacks against computer assets. In this paper, we introduce an original system whose purpose is to mitigate the benefits an attacker can expect from scanning a targeted network. In contrast to more traditional approaches, we propose to act a priori against scanning activity by continuously obfuscating the appearance of the targeted network through the combination of various simple mechanisms (i.e. random connection dropping and traffic forging). Moreover, we propose a method to immediately penalize hosts sending seemingly suspicious traffic to the targeted network while maintaining decent connectivity to cope with "false positives".

1 Introduction

Over the past two decades the Internet and underlying TCP/IP computer networks have expanded to the point that nowadays a considerable number of organizations depend on them to conduct business (e.g. electronic commerce). In the process, they naturally attracted the attention of malicious people who started to devise ways of compromising and exploiting resources connected to the Internet for profit. One way for attackers to locate potentially interesting resources is to directly scan the network, sending probes to a set of destinations in the IP address space and watching for replies. Actually, several dedicated tools exist to automate that task, thus enabling attackers to easily and quickly process large portions of networks. Skillful attackers may even write self-replicating programs (i.e. worms) that autonomously scan for new vulnerable targets and directly try to compromise the ones they find. Eventually, malicious scanning activity now constitutes a significant portion of the traffic exchanged over the Internet [19].

Surprisingly, despite its apparent importance, little seems to be done specifically against that activity. This situation is probably related to the fact that there is no real consensus on the harmfulness of unsolicited network scans as they usually do not inflict any damage per se and are indeed difficult to link to actual attacks. System administrators may then be tempted to simply ignore them... Nevertheless, several researchers consider unsolicited network scans as a threat and thus proposed mechanisms to protect networks against them.

Most of the currently available mitigation systems are based on automated detection algorithms that determine highly probable scanning sources; further traffic from those sources being subsequently denied. That approach is reminiscent of what is done with Network-based Intrusion Detection Systems (NIDSs) to deal with actual attack attempts. However, we believe it to be somehow inappropriate to deal with scanning activity: for instance, even with state of the art algorithms, accurate early detection of scanning sources is still problematic and therefore upon detection it is generally too late to prevent scanners from collecting the pieces of information they were looking for.

In this paper, we introduce an original system for network scan mitigation. In contrast to the above-mentioned prevalent approach, we tried to focus more on the specificities of scanning traffic, including its purpose. We designed our system to continuously reduce the pertinence of the pieces of information an attacker can gather by scanning a targeted portion of network, thus aiming directly at the raison d'être of unsolicited scans. To achieve that, we propose to make the network portion we ought to protect behave uniformly (i.e. to make it respond in a similar fashion no matter what traffic it receives). That uniform "look and feel" is produced by combining various traffic manipulation mechanisms. And, for increased mitigation efficiency, we propose to bias the way incoming traffic is processed based on basic trust/distrust considerations (i.e. "selective deception").

The rest of the paper is organized as follows. In Section 2, we introduce the principal network scanning techniques along with a selection of previously proposed mitigation techniques. In Section 3, we describe the proposed system and discuss its design and expected capabilities. Finally, in Section 4, we conclude on our proposal and expose some leads regarding our future work.

2 Background

2.1 Scope

As suggested earlier, we restrict our discussion to TCP/IP networks. Furthermore, we focus our attention on scanning activity occurring at the network and transport layers, the privileged scenario being remote scanners trying to find out active hosts on a target network. So our system is not meant to cope with local scanning activity (e.g. ARP scanning) and we will neglect scanning activity occurring at upper layers. In practice, we will mainly deal with the IP, ICMP, TCP and UDP protocols [3].

2.2 TCP/IP Network Scanning

Most of the TCP/IP network scanning techniques available in the wild are actually well understood and widely documented. This is probably due to their dual-use: for malicious purposes by attackers, but also for management and security assessment purposes by system administrators. Additionally, their basics have not evolved much over recent years. For instance, the survey written by De Vivo et al. almost a decade ago [5] is still fully accurate. In this section, we will briefly introduce the main scanning techniques that we try to mitigate with our proposal. For more detailed information, the reader can notably refer to the documentation of the Nmap tool [11].

When scanning a TCP/IP network, attackers are basically interested in knowing what hosts are up, the list of their open TCP/UDP ports (i.e. ports on which processes are listening for incoming data), and to a smaller extent, their Operating System (OS). ”Valid” open ports are of particular interest as they constitute potential entry points for subsequent malicious activity. The most straightforward way to obtain such pieces of information is to actively probe the targeted hosts by sending them packets compliant with the various TCP/IP protocol specifications and analyze the associated replies. For instance, to test if a host is up, one can try to send it ICMP echo requests and check whether he receives a reply or not (i.e. ”Ping scanning”). Similarly, to test if a TCP port is open, one can simply try to establish a regular connection to it by performing a complete three-way handshake [13]. That connection attempt either fails or succeed, thus revealing the status of the port.

A scanner may also try to deviate from the protocol specifications in order to gain stealth or refine is scanning results. For instance, compared to the previously described TCP port scanning technique, he could choose to only send a SYN segment to the targeted port to initiate a connection but never send the following segments necessary to fully establish it (i.e. half-open connection, ”TCP SYN scanning”). The answer to that initial SYN segment is indeed enough to guess the status of the port: a RST segment usually means that the port is closed, a SYN/ACK segment that the port is open and, no response that the port is probably protected by a packet filtering mechanism. Other TCP port scanning techniques generally involve the use of packets specially crafted to exploit very subtle parts of the TCP protocol specification (e.g. ”FIN”, ”Xmas Tree” and ”Null” techniques [5]). Regarding UDP ports, active probing with UDP segments can be used to try to determine their status but compared to TCP, it is much more difficult for a scanner to get a ”precise answer” (cf. open UDP ports rarely reply).

Finally, the OS of a host can be inferred by matching the characteristics of the packets it sends to ”signatures” derived from known OSs. Recent works on active fingerprinting show that it is possible to get an accurate estimation of the sought OS with a very few probes [7]. For instance, the SinFP tool [1] relies only on three or less probes sent to an open TCP port, the probes all being ”standard” TCP segments.

2.3 Detecting Scanners?

Most Network-based Intrusion Detection Systems (NIDSs) embed algorithms to detect scans [17]. Primitive algorithms were based on per-source counters: they basically triggered an alert if a source was trying to reach more than a predefined number of different ports or of different hosts during a specified time frame (e.g. Snort [15]). Jung et al. later proposed an improved detection algorithm [9] that greatly reduces the number of different hosts that a source can probe before being successfully flagged as a scanner (around 4 or 5 in practice). Yet, even with such detection mechanisms, willing attackers are still able to gather a fair amount of information about networks they decide to target. In particular, Kang et al. showed that an attacker controlling a reasonable number of colluding scanning sources could defeat Jung et al.’s proposal [10] and, with the present botnet phenomenon [12], such attackers are actually very likely to surface (if they have not already). Another issue is that in most cases, scanners can avoid detection by sufficiently spacing out their probes. Eventually, the use of information visualization techniques (e.g. InetVis [18]) provides a way to detect such slow scans but real-time response is then impaired.

2.4 Defense Mechanisms

System administrators usually respond to network scans by denying further traffic from identified scanning sources. To do so, they principally rely on firewalls. Initial firewalls were limited to simple packet matching criteria (e.g. their protocol, their source/destination IP addresses, etc.), packets being then either accepted or rejected according to predefined static rules. Nowadays, most firewalls are also stateful, they can keep track of connections passing through them and thus recognize whether a packet belongs to an already established connection or not. Rules are no longer solely static but can as well be created dynamically according to previous states or external information (i.e. reactive firewalls).

Those extended capabilities enable system administrators to develop elaborate dynamic filtering schemes to protect their networks. A common example is the automated blacklisting of source addresses based on the output of NIDSs. Still, that scheme is vulnerable to IP spoofing attacks: by forging intentionally suspicious network packets with fake source IP addresses, attackers can trigger the insertion of these addresses in such automated blacklists. They can then selectively prevent genuine users from accessing the targeted network or overload the blacklists with wasteful entries (i.e. Denial of Service (DoS) attacks). IP spoofing can also be exploited to evade defense mechanisms: attackers can try to conceal their probes by simultaneously sending similar spoofed packets. Moreover, by cleverly making use of spoofed packets it is even possible to bounce a scan off a random idle host (i.e. ”Idle scanning” [11]). This technique enables attackers to avoid identification as the scan appears to be coming from the selected idle host but has a few drawbacks (e.g. it generally consumes much more time than ”direct” scanning techniques). IP spoofing can be prevented by ingress filtering mechanisms [6] however these mechanisms are yet to be widely deployed.

IP spoofing if a fine example of deception for attacking purposes, but deception can also be used for defense [4]. The most widely used deceptive tools for defense are probably honeypots [14]. The term ”honeypot” actually designates a broad range of tools that basically act as traps for attackers. A honeypot is usually an otherwise unused resource (i.e. no production value) that is set in a network and ”waits” for activity. As it is ”unused”, most of the activity that it receives is actually malicious (or at least abnormal). Consequently, honeypots are mainly used for detection and information gathering. For instance, some honeypots can mimic services commonly targeted by attackers and therefore deceive them into performing complex attacks while being monitored.

Another noteworthy defense mechanism when dealing with network scans is packet scrubbing (also referred to as packet normalization). It consists in eliminating ”ambiguities” that exist in TCP/IP traffic: some parts of the TCP/IP protocol specifications can be interpreted differently and differences exist in the way these specifications have been implemented. For instance, packet scrubbers reassemble fragmented packets and drop TCP segments that have invalid or inappropriate flag combinations. Therefore, packet scrubbers can hinder OS fingerprinting attempts (as they make the traffic generated by different TCP/IP stacks indistinguishable [16]), and thwart TCP port scanning techniques that rely on unusual packets. Smart et al. [16] proposed an initial packet scrubber implementation and some normalization functionalities were also later added to PF [8].

3 MitiBox

3.1 Incentives and General Description

End-to-end connectivity is one of the core principles the Internet was built upon. It basically stipulates that communication protocols should be designed as to store any necessary state information at the endpoints and not in between (i.e. not in the network). A direct consequence of that principle is that any host connected to the Internet can virtually send traffic to any other connected host. That condition makes scanning activity very difficult to suppress as it is somehow intrinsically linked to the way the Internet is operating: any connected host is a potential scanner. One solution would be to abandon the end-to-end connectivity principle, as suggested for instance by Ballani et al. with their ”default-off” paradigm [2]. However, such approach would require the Internet infrastructure to be heavily modified and therefore we believe it to be quite impracticable for the time being.

If we assume that we cannot eliminate scanners and prevent them from sending probes to the internal network (i.e. the portion of network that we ought to protect), the most obvious strategy would be to try to detect those probes and discard them before they reach their intended targets. However, as previously discussed (cf. Section 2.3), the effectiveness of that approach is limited as the first probes sent by a source are usually allowed to slip through (i.e. until that source is positively identified as a scanner). And alas, the associated responses are already leaking out precious information...

We believe that that issue could be overcome by adopting a slightly more aggressive approach: all incoming traffic to the internal network is to be initially considered as probing traffic and treated as such until proved otherwise. Based on that presumption, we propose to put in place mechanisms at the border of the internal network that continuously obfuscate its appearance. Then, how should the internal network ”behave” to continuously deceive potential scanners while still enabling genuine hosts to access the services it provides? We propose to implement the following configuration: all malformed incoming traffic is to be dropped; the remaining traffic is to either be dropped or to be replied to with equal probability $(1/2)$, replies being forged when needed. By doing so, our goal is to make the internal network in its entirety behave uniformly so that all probing traffic is replied in a similar fashion. Indeed, perfect uniformity might be hard to attain but we believe that even approximate uniformity can effectively disrupt scanners.

In addition, we propose to evaluate sources sending traffic to the internal network based on their initial activity. More precisely, we propose to assign a trust level to each source based on the destination it first tries to reach. If that destination is deemed legitimate, the source is flagged as trusted, otherwise it is flagged as distrusted. The trust level assigned to a source is then used as a factor that influences the processing of subsequent incoming traffic from that source: traffic from a source flagged as distrusted is to be prevented from reaching the internal network (i.e. diverted or dropped). Indeed, such scheme presupposes a way to evaluate destinations. In practice, the number of services that should be made available to the outside of a network is usually quite low and the details of these services known to system administrators. In that case, it is possible to precompile an exhaustive static list of all the legitimate destinations on the network (i.e. list of IP addresses and TCP/UDP ports). Yet, for networks with rather lax access policies, precompiling such static list can become quite difficult. One could then rely on a dynamic list filled based on the actual response internal hosts previously gave to incoming traffic (e.g. by monitoring outgoing traffic), or based on administrative scans of the network. Naturally, such dynamic list would have to be periodically refreshed to reflect the composition of the network. In the remaining of the paper we assume that we can construct/obtain a list of destinations on the internal network that are likely to be legitimate.

Ideally, genuine external hosts should not be penalized by the proposed scheme as they usually know a priori the location of the services that they want to use and thus are not likely to be flagged as distrusted. Nevertheless it may happen, for instance due to misconfiguration issues or spoofing attacks. To deal with such cases, we propose to make the trust level associated with observed sources gradually return to a neutral level (i.e. untrusted).

3.2 Structure

The proposed system is based on several modules, each providing a specific set of functionality:

- the "Scrubber" module: it drops abnormal traffic (i.e. not compliant with the TCP/IP protocol specifications) and smooths the remaining traffic, eliminating ambiguities that could disrupt other modules.
- the "Tracker" module: it tracks protocol states, making it possible to determine whether a packet belongs to a previously established communication session between two hosts and thus to apply the same treatment to all packets that make up that communication session. By communication session, we indeed refer to TCP connections but also to so-called UDP connections (hence we will abusively employ the term "connection" for "communication session"). For protocols without explicit connection starting/ending packets (e.g. UDP), the "Tracker" module keeps track of how long it has been since it witnessed a packet matching the associated state and if that duration exceeds a timeout the state is cleared (i.e. behavior similar to most stateful firewalls). The "Tracker" module also records what processing is to be applied to the packets of each connection it tracks (i.e. as determined by the other modules) and carries that processing out.
- the "Probability" module: it randomly drops a fraction of the connection attempts it receives.
- the "Trust" module: it records and manages the trust level associated with each external source. Then, according to that information, it determines whether the connection attempts under scrutiny should be diverted to the "Reply" module or passed on to the "Ruleset" module for further inspection. External sources are originally considered as untrusted. However, once one of them tries to reach an internal host, that source is assigned an initial trust level by the "Ruleset" module: either "trusted" or "distrusted". The "trusted" status is to last only for the connection whose establishment triggered it. All packets associated with that connection are then to be delivered to their intended destination (cf. "Tracker" module). The "distrusted" status is to last for a duration T_1. All connection attempts from a "distrusted" source are to be diverted to the "Reply" module. After T_1, an initially "distrusted" source is then to be considered as "partially distrusted" and assigned a probability P that progressively decreases over time. The connection attempts from a "partially distrusted" source are to be diverted to the "Reply" module with the probability P or passed on to the "Ruleset" module with the probability $(1 - P)$. Figure 1 explicits the transitions between the various trust levels for the case of three discrete "partially distrusted" levels.
- the "Ruleset" module: it specifies which destinations on the internal network are to be considered as

legitimate. Based on that, it determines whether the sources of the connection attempts under scrutiny should be subsequently distrusted or not, and processes the associated traffic accordingly.

- the ”Reply” module: it replies in a credible and indistinguishable fashion to the traffic that has been redirected to it. Its behavior is actually very similar to low interaction client honeypots [14]. However, if honeypots are often used to gather information on attackers (cf. Section 2.4), the ”Reply” module is more about hiding information from them [20].

3.3 Traffic Processing

The modules previously introduced are to process all traffic exchanged between the internal network and the outside. The outgoing traffic (i.e. connections initiated from the internal network) is only to go through the ”Scrubber” module and the ”Tracker” module as we assume the internal network to be trusted. As for the incoming traffic, its processing can roughly be divided into four main steps, as follows (cf. Figure 3):

1. Similarly to the outgoing traffic, the incoming traffic is first to be processed by the ”Scrubber” module and the ”Tracker” module: invalid traffic is dropped and packets corresponding to registered connection attempts are applied the processing that was previously selected for them (cf. the next three steps).

2. After the first step, remaining packets to be processed should all be connection opening packets. These connection attempts are to be indiscriminately dropped with a probability of $1/2$. An independent decision (i.e. to drop or not to drop) is to be made for each new connection attempt and hence all packets related to the corresponding connection are to later undergo the same treatment (cf. decision stored in the ”Tracker” module).

3. The sources of the remaining connection attempts (i.e. not dropped in the previous step) are then to be checked against a list of currently distrusted source addresses. In case of a positive match, the corresponding connection is considered for diversion. The decision to divert a connection or not is made based on the current level of distrust of its initiator.

4. Finally, the destinations of the remaining connection attempts (i.e. not diverted in the previous step) are to be checked against a ruleset that specifies legitimate destinations on the internal network. If a connection attempt is targeting a legitimate destination, it is to be let through (i.e. source temporary affected the ”trusted” status). Otherwise, if a connection attempt is targeting an illegitimate destination, it is to be diverted and its source is to be added to the list of currently distrusted addresses (i.e. source affected the ”distrusted” status).

Figure 2 summarizes how packets are handled. As packets related to dropped connection attempts are to be dropped no matter the trust level of their source address, their case is omitted on the figure for clarity.

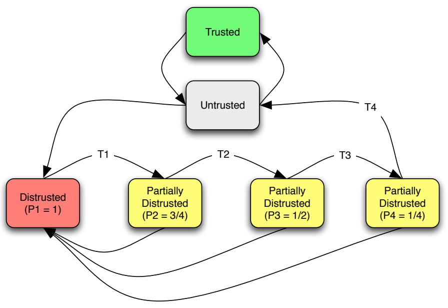
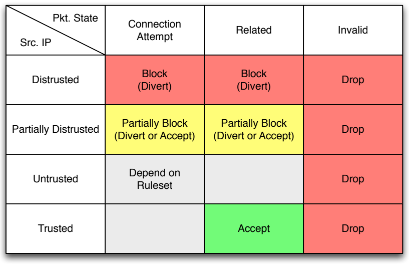

Trusted
Untrusted
T4
T1
T2
T3
Distrusted (P1 = 1)
Partially Distrusted (P2 = 3/4)
Partially Distrusted (P3 = 1/2)
Partially Distrusted (P4 = 1/4)

Figure 1: Relations between the various trust levels (for three discrete ”partially distrusted” levels)



Pkt. State / Src. IP	Connection Attempt	Related	Invalid
Distrusted	Block (Divert)	Block (Divert)	Drop
Partially Distrusted	Partially Block (Divert or Accept)	Partially Block (Divert or Accept)	Drop
Untrusted	Depend on Ruleset		Drop
Trusted		Accept	Drop

Figure 2: Processing of incoming packets based on their state and the trust level of their source IP address.

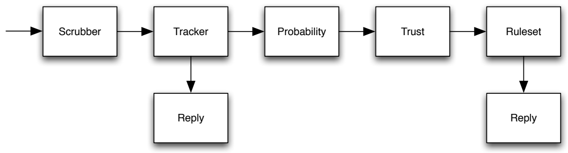


Figure 3: Chain of modules associated with the processing of incoming traffic.

3.4 Discussion

We expect the proposed system to mitigate most of, if not all, the network scanning techniques introduced in Section 2.2. Techniques based on non-standard packets and TCP/IP stack fingerprinting are hindered by the combination of the ”Scrubber” module and the ”Tracker” module. Connection attempts (and ICMP echo requests) are consistently either dropped or replied to, by either authentic internal hosts or the ”Reply” module. Therefore, finding interesting hosts is made harder for potential attackers. Obviously, it depends on the abilities of, the ”Scrubber” module to smooth traffic and of, the ”Reply” module to mimic authentic hosts. However, considering the tools currently available for that effect, we believe that a satisfactory level of deception is achievable. Moreover, with our system, the probability for an attacker to actually reach an authentic host by means of ”direct” scans is fairly low. If he blindly tries to connect to random destinations of the internal network, he will most probably be flagged as ”distrusted” and his subsequent attempts diverted to the ”Reply” module. And even if he initially guesses ”correctly”, his attempts may be dropped by the ”Probability” module. By bouncing his scans off several idle hosts (cf. Section 2.4), an attacker could avoid being flagged as ”distrusted” and increase his chances to hit authentic hosts. However, such approach would consume much more of his time and he would still have to face the other deception mechanisms (i.e. the ”Probability” module and, the need to differentiate between authentic hosts and the ”Reply” module).

4 Conclusion and Future Work

In this paper, we proposed a system for network scan mitigation based on the combination of various simple mechanisms. The originality of our system resides in the fact that with it, mitigation is continuous as opposed to more traditional approaches where it is only triggered intermittently (i.e. by automated detection algorithms). By doing so, our system is expected to offer a more comprehensive protection against scanners. Additionally, by design, most of the currently available scanning techniques should see their effectiveness severely reduced (e.g. consumption of far more resources to acquire accurate information or failure to do so). As for future work, we plan to complete our implementation of the proposed system and to deploy it in a real environment in order to evaluate its practical efficiency and performance.

5 Acknowledgments

The author would like to thank Yoshiaki Hori, Pascal Jinkoji, Heejo Lee, Frédéric Majorczyk and Antoine Brodin for their valuable comments. Partial support for this work was provided by the National Institute of Information and Communications Technology (NICT), Japan (”Research and Development for Widespread High-Speed Incident Analysis” project).

References

[1] AUFFRET, P. SinFP. `http://www.gomor.org/bin/view/Sinfp/`.

[2] BALLANI, H., CHAWATHE, Y., RATNASAMY, S., ROSCOE, T., AND SHENKER, S. Off by Default! In *Proceedings of the 4th workshop on Hot Topics in Networks (HotNets-IV)* (New York, NY, USA, November 2005), ACM SIGCOMM.

[3] BRADEN, R. RFC 1122: Requirements for Internet Hosts – Communication Layers, 1989.

[4] COHEN, F. A Note on the Role of Deception in Information Protection. *Computers & Security 17*, 6 (1998), 483–506.

[5] DE VIVO, M., CARRASCO, E., ISERN, G., AND DE VIVO, G. O. A Review of Port Scanning Techniques. *SIGCOMM Computer Communication Review 29*, 2 (1999), 41–48.

[6] FERGUSON, P., AND SENIE, D. RFC 2827: Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing, 2000.

[7] GREENWALD, L. G., AND THOMAS, T. J. Toward Undetected Operating System Fingerprinting. In *Proceedings of the 1st USENIX workshop on Offensive Technologies (WOOT'07)* (Berkeley, CA, USA, 2007), USENIX Association, pp. 1–10.

[8] HARTMEIER, D. PF. `http://www.openbsd.org/faq/pf/`.

[9] JUNG, J., PAXSON, V., BERGER, A. W., AND BALAKRISHNAN, H. Fast Portscan Detection using Sequential Hypothesis Testing. In *Proceedings of the 2004 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 211–225.

[10] KANG, M. G., CABALLERO, J., AND SONG, D. Distributed evasive scan techniques and countermeasures. In *Proceedings of the 4th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'07)* (Berlin, Heidelberg, 2007), Springer-Verlag, pp. 157–174.

[11] LYON, G. F. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, USA, 2009.

[12] MCCARTY, B. Botnets: Big and Bigger. *IEEE Security and Privacy 1*, 4 (2003), 87–90.

[13] POSTEL, J. RFC 793: Transmission Control Protocol, 1981.

[14] PROVOS, N. A Virtual Honeypot Framework. In *Proceedings of the 13th conference on USENIX Security Symposium (SSYM'04)* (Berkeley, CA, USA, 2004), USENIX Association, pp. 1–1.

[15] ROESCH, M. Snort. `http://www.snort.org/`.

[16] SMART, M., MALAN, G. R., AND JAHANIAN, F. Defeating TCP/IP Stack Fingerprinting. In *Proceedings of the 9th conference on USENIX Security Symposium (SSYM'00)* (Berkeley, CA, USA, 2000), USENIX Association, pp. 17–17.

[17] STANIFORD, S., HOAGLAND, J. A., AND MCALERNEY, J. M. Practical Automated Detection of Stealthy Portscans. *Journal of Computer Security 10*, 1-2 (2002), 105–136.

[18] VAN RIEL, J.-P., AND IRWIN, B. InetVis, a Visual Tool for Network Telescope Traffic Analysis. In *Proceedings of the 4th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa (Afrigaph'06)* (New York, NY, USA, 2006), ACM, pp. 85–89.

[19] YEGNESWARAN, V., BARFORD, P., AND ULLRICH, J. Internet Intrusions: Global Characteristics and Prevalence. In *Proceedings of the 2003 ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'03)* (New York, NY, USA, 2003), ACM, pp. 138–147.

[20] YUILL, J., DENNING, D., AND FEER, F. Using Deception to Hide Things from Hackers: Processes, Principles, and Techniques. *Journal of Information Warfare 5*, 3 (2006), 26–40.