A Technique for Network Topology Deception

Samuel T. Trassare
Naval Postgraduate School
Email: sttrassa@nps.edu

Robert Beverly
Naval Postgraduate School
Email: rbeverly@nps.edu

David Alderson
Naval Postgraduate School
Email: dlalders@nps.edu

***Abstract*—Civilian and military networks are continually probed for vulnerabilities. Cyber criminals, and autonomous botnets under their control, regularly scan networks in search of vulnerable systems to co-opt. Military and more sophisticated adversaries may also scan and map networks as part of reconnaissance and intelligence gathering. This paper focuses on adversaries attempting to map a network's *infrastructure*, i.e., the critical routers and links supporting a network. We develop a novel methodology, rooted in principles of military deception, for deceiving a malicious traceroute probe and influencing the structure of the network as inferred by a mapping adversary. Our Linux-based implementation runs as a kernel module at a border router to present a deceptive external topology. We construct a proof-of-concept test network to show that a remote adversary using traceroute to map a defended network can be presented with a false topology of the defender's choice.**

I. Introduction

Networks are probed thousands of times a day for vulnerabilities and points of access [1]. Such scans may originate from hackers and cyber criminals, or more sophisticated military adversaries performing reconnaissance and intelligence gathering. This paper focuses on adversaries attempting to map a network's *infrastructure topology*, i.e., the critical routers and links supporting a network.

The intuitive response to such near-constant probing is to selectively deny the adversary's probes from entering the network. Instead of merely denying the adversary a vantage into a defended network, we examine the value to be gained in employing the principle of military deception. Our novel technique for network topology deception presents an adversary with a false network topology that purposefully conceals and disguises the underlying true topology. Such deception can frustrate an adversary's ability to gain usable intelligence from a target, and thus prevent exploitation.

Our research focuses on network mapping adversaries performing active probing. The primary tool employed for such probing is traceroute [2]. The intelligence gained from traceroute probing allows an attacker to gain valuable insight into which nodes in the network would, if disabled by attack, yield the greatest overall impact (e.g., network disruption, or other objective). We demonstrate the ability to present the illusion of any arbitrary topology of the defender's choice, ranging from random to false topologies purporting to be much more resilient than they are in actuality.

We develop a working Linux-based implementation of our technique as a proof-of-concept. The implementation relies on the inherent lack of authenticity in responses to traceroute probes (i.e., ICMP [3]) to provide deception in which the *outward appearance* of a network topology is altered. The topological deception may provide the perception of a network that resembles, or completely disguises, the underlying true network by varying attributes such as nodes, node count or the redundancy and diversity of links between nodes.

For instance, the outward topology presented to an attacker may be chosen to protect high-value nodes or links within the network. Thus we may cause the adversary to take specific actions, such as attacking highly fault-tolerant nodes that appear weak, or avoiding weak nodes that appear highly fault-tolerant. The methodology accommodates any true input topology, while the deceptive topology can be modified easily and frequently to further confound the adversarys efforts to identify vulnerabilities in the network infrastructure.

We evaluate our technique on a test network to demonstrate its efficacy. The execution of our Linux kernel module implementation is shown to work within a synthetic network topology with an adversary probing the path to a publicly available service (a web server) whereby the attacker infers the deceptive (false) route. Our experiments show that a defender using our technique can successfully deceive a traceroute probe, the first in a sequence of steps to mount a credible deception scheme against an adversary. This initial work demonstrates the potential of the technique and the utility of network topology deception.

The remainder of this paper discusses related work (§II), the design of our topological deception implementation (§III) and results (§IV). Before concluding we discuss the implications of our topological deception in defending DoD networks.

II. Background

There are at least two methods of employing deception in cyberspace. The first is through *obfuscation*, in which defensive software tools mask identifying attributes of a host computer to prevent "fingerprinting" [4]. The second method, the use of honeypots, lures the adversary into exploiting seemingly vulnerable computers which are, in reality, programs running on a host computer expressly to deceive an adversary into exposing his or her tactics [5].

Whaley [6] describes deception in two categories: hiding the real, or *dissimulation*; and showing the false, or *simulation*. Dissimulation is that part of a deception that is concealed from the adversary. Simulation is the overt part of a deception presented to the adversary. For example, consider an implementation of TCP/IP in which the generated traffic contains artifacts that allow their host OSs to be identified [7]. Smart *et al.* [8] describe a scrubber that obfuscates the behavior of the TCP/IP stack to prevent host operating system identification (dissimulation).

In simulation research, the work presented by Frederick [9] explores the state of the art in honeypots by testing the low-interaction honeypot software on an Internet-facing network without the protection of a firewall. Honeypots perform by Whaley's definition of simulation by providing the illusion of an existing network resource.

To the best of our knowledge, our work is the first to explore the notion of network topology obfuscation. The research presented herein employs both dissimulation and simulation. Perhaps most closely related to our work is that of recent "moving target defense" initiatives [10] that attempt to increase the complexity and cost of an adversary's attack by shifting and changing over time.

Yuill *et al.* describe three discovery processes by which an adversary gains knowledge of a network. Those processes are direct observation, investigation, and learning from other people or agents [11]. Our work attempts to impair the adversary's ability to perform direct observation by subverting the active probing tool, traceroute.

III. Methodology

Our deception focuses on deceiving an adversary's traceroute [2] probe. Traceroute discovers the forward path (sequence of router interfaces) of data packets from the vantage on which it is run to a specified destination. By artificially manipulating each probe packet's Time-to-Live (TTL), traceroute elicits ICMP [3] Time Exceeded messages from routers along the path. Augustin *et al.* [12] provide an in-depth treatment of traceroute and analyze the potential for false topological inference due to traceroute anomalies. Exploiting traceroute is the focus of this work. Crucially, the path reported to the prober is a function of the source IP addresses of these Time Exceeded messages from the routers.

Our primary observation is that ICMP lacks authenticity and integrity, thus the source IP address may take on any value. As a result, there are many possible strategies to implement deception. In this section, we discuss two deception options, then describe our real-world Linux implementation.

A. Random Deception

We consider two methods of deceiving a traceroute probe. In the physical domain, an Electric Warfare (EW) suite has the capability to either saturate the Radio Frequency (RF) spectrum in response to an adversary's radar sweep, or it may subtly alter the attributes of the adversary's radar sweep response so as to disguise the actual distance between the adversary and target.

Likewise, a deceptive response to a traceroute probe may be "noise" in the form of ICMP Time Exceeded messages with randomly generated source IP addresses for each reply packet[1]. Such a deception may prove valuable in frustrating automated traceroute probes as with the method employed by LaBrea [13]. Or, the deceptive response may be a plausible and complete path to the target. Both methods were implemented during our experimentation. We used a random number generator to generate random source IP addresses, what we considered to be the "noise."

Returning random IP addresses to an adversary may be deterministic on a per-source basis, or may return a random path even for sequential invocations of traceroute. Clearly, such random behavior is detectable by sophisticated adversaries. However, there exists value in simply frustrating mapping attacks and keeping the adversary in an intelligence gathering phase [10].

B. Intelligent Deception

A more sophisticated deception results in a "believable" topology. We consider one particular operational goal: to take the true network topology as an input such that the resulting deceptive topology protects critical high-value infrastructure nodes.

An important consideration was identifying what constitutes a "high-value" node. A high-value node is one whose exploitation would cause the largest impact to the network. There are many metrics for evaluating the high-value targets in a network infrastructure. For this work, we use the well-established graph theoretic notion of betweenness centrality in determining a node's value.

Betweenness centrality (C_B) is a node or link measure that represents how important the node or link is to the connectivity of the rest of the network. Freeman [14] describes the notion of point centrality as a structural property of betweenness, explaining, "a point in a communications network is central to the extent that it falls on the shortest path between pairs of other points." A node with high C_B has greater ability to relay or withhold communication between the pairs of nodes it connects compared to other nodes on the path between the same pair. It follows then that, should a node with high C_B be disabled by attack or other action, its inability to relay communication greatly impacts the pairs of nodes it connects.

As an exemplar of what is possible with our topological deception, we identify the node with the largest C_B as the "most vulnerable node." We then implement a strategy that presents a false topology that minimizes the maximum betweenness by adding a single false edge to the true topology. We explore the potential for more complex deception in §V, but limit our initial approach to a single false link[2].

We test two algorithms that determine the false link in deceptive topology. The first greedy algorithm is simple: find the two nodes in the network with the lowest C_B and connect them. The second approach is to exhaustively enumerate all possible networks, adding one link between every pair of nodes in the network, and returning the model of the network with the lowest maximal C_B. While the exhaustive approach is optimal, it is also inefficient: for n nodes, the algorithm must compute C_B for $\binom{n}{2}$ potential networks. We leave developing a more efficient algorithm to future research.

The greedy approach produced results that were equal to that of the exhaustive approach 50% of the time for 100 randomly generated Watts-Strogatz [15] input networks. Each input network was created with 10 nodes with each node

[1] Assuming that the original TTL-limited probes are prevented from reaching actual routers, or that the true ICMP responses are blocked.

[2] This is not an inherent limitation, but was chosen to scope the problem.

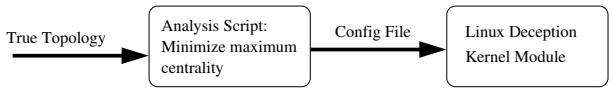

Fig. 1: High-level operation of the implementation: the true topology is analyzed for weaknesses (nodes with high centrality) in order to drive the intelligent deception.

Listing 1: Sample kernel configuration file specifying deception

```
options deceptiveroute hop_addr =
192.168.0.2, 192.168.4.2, 192.168.5.2
```

connected to three nearest neighbors in ring topology with p, the probability of rewiring each edge, equal to 0.2.

The exhaustive algorithm generated graphs that frequently showed lower C_B than the greedy algorithm. We therefore use the exhaustive algorithm in our implementation.

We developed a Python script that takes an input file describing a true network topology. The input file contains a list of all nodes in a network, each node's IP address and each of the links in the network. The script then performs an exhaustive search of the graph, using the NetworkX [16] library, testing the resulting overall C_B after adding a single new link between each pair of nodes in the graph. The output of the script is the input to our deceptive kernel module that achieves a deception whereby the maximum C_B is minimized. Figure 1 provides a high-level overview of the operation of the intelligent deception. The resulting configuration file is shown in Listing 1 where `deceptiveroute` is the name of the module and `hop_addr` is the name of the variable to be initialized with deceptive IP addresses).

C. Kernel Module Implementation Details

The heart of the deception lies in the implementation of a Linux kernel module. Table I lists several important variables that are initialized when the kernel module is loaded. The variable `arr_argc` contains a count of the number of elements passed to `hop_addr` and represents the maximum number of hops provided in the deceptive route. The configuration file used as an input to the kernel module (`hop_addr`) describes a topology, on a per-destination basis, that lowers the perceived C_B of the most vulnerable node, thus making it a less attractive target for attack.

The ICMP Time Exceeded replies from the routers are IP packets, and therefore contain TTL values themselves. The round-trip time and the TTL of a packet, being loosely correlated, should reinforce the deception. Different TCP/IP stacks use different initial TTL values when originating packets. The recommended default TTL value given by IANA [17] is 64. However, many vendors choose different values such as 32, 128 and 255. The Cisco routers used in our experiment have a default TTL of 255. The DEFAULT_TTL variable in the reply message may be set to one of these values if there is a desire to make the deceiving node appear to be a device from a different vendor.

In order to deceive an external traceroute to the specified destination (`destination_ip`) by returning a path

hop_addr	Initialized from configuration file (e.g., Listing 1)
arr_argc	Initialized to the number of IP addresses in hop_addr
preserve_ttl	Initialized to zero. Stores incoming packet's TTL from start of prerouting hook until end of postrouting hook
DEFAULT_TTL	Initialized to 65 per RFC 1700 with 1 added to account for programming logic
router_ip	192.168.0.2
destination_ip	192.168.5.2

TABLE I: Variables initialized at kernel module load.

dependent on the configuration file, we use the prerouting and postrouting hooks from the Netfilter [18] packet filtering package. The prerouting hook can intercept a packet upon arrival from the network interface card (NIC), before further kernel processing. Conversely, the postrouting hook can intercept a packet just before it leaves the host computer, after "normal" kernel processing. Figure 2a provides our prerouting pseudocode, while Figure 2b provides postroute pseudocode.

In the prerouting hook, the packet is inspected to determine if it is a traditional UDP-based traceroute[3] probe toward the `destination_ip`, i.e., UDP destination port 33434-33534 (as defined by IANA).

The deceptive hop that the kernel module chooses to send in response to the adversary's traceroute depends on the TTL value of the incoming packet. Traceroute sends packets with incrementally increasing TTL to identify every node in the path to a destination, thus the first packet to arrive at a border router of a network will always arrive with TTL=1. Therefore, UDP traceroute probes that arrive at the intelligent router with TTL=1 are assumed to be the first packet in a set of traceroute packets. For the first packet, no deception is returned to the adversary. The intelligent router truthfully replies to the adversary with an ICMP Time Exceeded message [3]. The kernel module enters its deception logic, but because the first IP address assigned to hop_addr is that of the router, no deception is actually made. This behavior is based on the fact that the deception begins after the point of ingress into a network where we envision the intelligent router being deployed.

When the second traceroute packet arrives (using UDP and a destination port range of 33434 to 33534) it has TTL=2. The reply to this packet is the first to be deceptively returned to the adversary. From the prerouting hook, the packet's TTL is saved in a temporary variable, `preserve_ttl`, to be accessed later in the postrouting hook. The packet's TTL is then set to zero. Doing so causes the packet to be processed by the host computer. In effect, the intelligent router is tricked into processing the packet as if it expired while on its way to the destination. The host prepares an ICMP Time Exceeded message to be sent to the traceroute origin (the adversary).

When the Time Exceeded message arrives at the postrouting hook, the source IP of the packet is the intelligent router. The postrouting hook therefore changes the source address to the first deceptive hop as defined by the configuration file.

Next, the TTL value of the new packet is decreased to reflect the metric the adversary expects for a hop that is more distant than the intelligent router. Here, the temporary

[3]While traceroute may use other transport protocols, we leave deceiving these for future work

```
Data: socket_buffer skb
Result: For input packet with TTL>1, set TTL=0
ip_header = get_ip_header(skb);
if ip_header→protocol == UDP then
    udp_header = get_udp_header(ip_header);
    if (33434 ≥ udp_header→dest_port ≤ 33434) AND ip_header→dest_address == webserver_ip then
        if ip_header→ttl == 1 then
            preserve_ttl = ip_header→ttl;
        else if ip_header→ttl > 1 AND ip_header→ttl < arr_argc then
            preserve_ttl = ip_header→ttl;
            ip_header→ttl = 0;
            ip_header→checksum == new_ip_checksum(ip_header, ip_header→length);
        else if ip_header→ttl == arr_argc then
            preserve_ttl = ip_header→ttl;
            ip_header→dest_addr = router_ip;
            ip_header→checksum == new_ip_checksum(ip_header, ip_header→length);
        end
    end
end
return ACCEPT_PACKET;
```

(a) Identify incoming traceroute probes and conditionally modify the TTL.

```
Data: socket_buffer skb
Result: Transmits a deceptive packet of type Time Exceeded or Port Unreachable
ip_header = get_ip_header(skb);
if ip_header→protocol == ICMP AND ip_header→src_addr == router_ip AND preserve_ttl > 0 then
    icmp_header = get_icmp_header(ip_header);
    if icmp_header→type == ICMP_TIME_EXCEEDED then
        ip_header→src_addr = hop_addr[preserve_ttl-1];
    else if icmp_header→type == ICMP_PORT_UNREACHABLE then
        quoted_ip_header = get_quoted_ip_header(ip_header);
        quoted_ip_header→dest_addr = webserver_ip;
        quoted_ip_header→checksum = new_ip_checksum(quoted_ip_header, quoted_ip_header→length);
    end
    ip_header→ttl = DEFAULT_TTL - preserve_ttl;
    ip_header→checksum == new_ip_checksum(ip_header, ip_header→length);
    delay_sending_packet(DELAY * preserve_ttl);
end
preserve_ttl = 0;
return ACCEPT_PACKET;
```

(b) Provide a deceptive Time Exceeded or Port Unreachable message to the adversary.

Fig. 2: Pseudocode for pre- and postroute hooks.

variable, `preserve_ttl`, and the `DEFAULT_TTL` value are used. Recall that the first packet sent in response to the traceroute scan is without deception and left the intelligent router with the default TTL=64. The second packet received at the intelligent router is originated by the adversary with a TTL one greater than the first packet received. For the second packet, the adversary expects to reach a machine or router that is one node more distant than the intelligent router. Thus the TTL of the reply packet should be one less than the previous packet. We preserve this behavior in the deception by using the inbound packet's TTL whose state is preserved in the prerouting hook. The TTL of the second outbound packet is set to `DEFAULT_TTL - preserve_ttl`. Following this step, `preserve_ttl` is set to zero to prepare it for the next packet.

Finally, the packet is delayed by a multiple of the original packet's TTL value to account for the additional time it should take for the packet to reach a hop that, again, is further away from the adversary than the intelligent router. The total delay imposed on each packet is 2 ms multiplied by the inbound packet's TTL. This deterministic delay is plausible, but future work will explore drawing from a probabilistic delay distribution. The functionality in the prerouting and postrouting hooks process each subsequent traceroute probe for which the deception requires a Time Exceeded message in reply.

In a typical traceroute, probing terminates when the packet reaches the destination if the destination replies with an ICMP Port Unreachable message. Our deception must be capable of providing deceptive paths of arbitrary length. In the prerouting hook, when the incoming TTL value matches the maximum number of hops (the value stored in `arr_argc`) required by the deception, the destination address of the packet is replaced by that of the host machine. In "local processes" the intelligent router creates the ICMP Port Unreachable message and passes the packet to the postrouting hook. In the postrouting hook, the same TTL manipulation is imposed. Lastly, an ICMP Port Unreachable packet contains as its data payload a portion (28 bytes) of the original packet (the "quotation") that induced the ICMP packet [19]. If the adversary's intended destination IP is not replaced in this field the deception is exposed. Thus when sending the ICMP Port Unreachable message, we ensure that the payload is correct by replacing the packet's source IP address with that of the adversary's destination IP (stored in `destination_ip`). The reply is then sent after imposing the delay constraints.

D. Testbed

Our implementation of topological deception is generalized to take as input any true network topology. However, for the purpose of illustration and testing, we use the topology of Figure 3 as our testbed. In this testbed, there is a single intelligent router implementing the deception, and a single web server acting as the destination target.

The realization of this testbed is via virtualization using the Graphical Network Simulator (GNS3) [20]. GNS3 runs actual

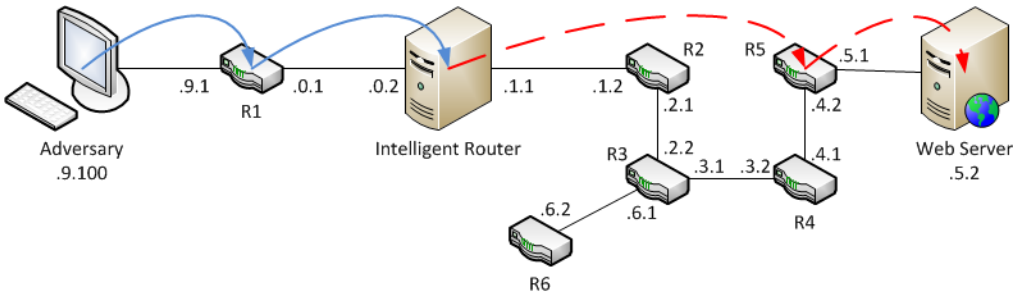

Fig. 3: Example topological deception: solid arcs represent the truthful route provided to the adversary in response to a traceroute probe. Dashed arcs represent the deceptive route supplied by the intelligent router. Node R3 has the highest C_B.

Listing 2: Traceroute results (prior to deception). Device labels from Figure 3 added for reference

```
traceroute to 192.168.5.2 (192.168.5.2),
30 hops max, 60 byte packets
 1   192.168.9.1   1.280  ms (R1)
 2   192.168.0.2   3.966  ms (Intelligent Router)
 3   192.168.1.2   5.997  ms (R2)
 4   192.168.2.2   10.097 ms (R3)
 5   192.168.3.2   12.135 ms (R4)
 6   192.168.4.2   14.330 ms (R5)
 7   192.168.5.2   16.109 ms (Web Server)
```

router images, and allows virtual machines (VMs) to interconnect with any arbitrary assemblage of links. This virtual testbed allows for easy experimentation by permitting links and hosts to be changed quickly without the overhead of assembling and reconfiguring candidate topologies in expensive hardware.

IV. Experiment Results

This section discusses results based on experiments with our candidate network topology in Figure 3 and our deceptive kernel module implementation.

As a baseline against which to assess the quality of our deception, we start with a standard Linux UDP-based traceroute to the destination web server in the example topology. Listing 2 shows the output of the traceroute command when the deception implementation is withheld. To make the output listing clearer, we limit the number of probes per TTL hop to one and include device labels matching the devices in Figure 3. For each hop, the IP address of the forward-facing interface of each node is shown in the traceroute output.

A. Random Deception

As alluded to previously, there are a range of possible deceptions. We begin by performing random deception to demonstrate the feasibility and promise of the approach. In this respect, the random deception is similar to radar jamming or the entrapment capabilities of LaBrea.

We test our ability to generate noise in which we return random IP addresses to an adversary's traceroute probe. The results of random deception are given in Listing 3. The deception returns Time Exceeded messages containing randomly generated source IP addresses, thus the adversary never receives the Port Unreachable message it expects and continues to probe until it reaches traceroute's default probe limit of 30 hops before quitting. Through experimentation we observe that our implementation periodically fails to respond to

Listing 3: Traceroute results with random deception. Output truncated to 9 hops

```
traceroute to 192.168.5.2 (192.168.5.2),
30 hops max, 60 byte packets
 1   192.168.9.1       1.039 ms
 2   132.65.218.87     3.996 ms
 3   240.184.140.169   3.935 ms
 4   247.10.122.16     4.178 ms
 5   153.55.189.76     3.956 ms
 6   255.253.22.13     4.126 ms
 7   112.52.193.63     3.942 ms
 8   *
 9   213.218.8.151     2.829 ms
     ...
```

incoming traceroute packets resulting in non-responsive hops being reported by traceroute. The non-responsive hops are denoted by the asterisks shown in the output listing in Figure 3. We leave resolution of the bug to future work, but note that it helps to reinforce the realism of the deception as many real routers do not generate ICMP Time Exceeded responses, or otherwise block ICMP.

B. Intelligent Deception

The true test network topology was fed as input to our Python script to find the node with the highest C_B and provide a false topology with an addition of a single new link that minimizes this maximum C_B. The output of the script is the kernel module configuration file of Listing 1 and describes the hops necessary, including the new link, to reach a particular destination (a web server in our test network). The new link provides the illusion of additional redundancy in the network whereby multiple paths can route traffic instead of a single critical node with the highest C_B.

With the kernel module deception running, we act as an adversary by performing a traceroute to the destination web server. The intelligent router, detecting the incoming probes, returns the prescribed deceptive route to the adversary.

In order to ensure that the deception does not impede other non-traceroute traffic, the browser on the adversary's machine is used to view the web pages provided by the web server. Correct operation of the deception is further determined by attempting to traceroute to devices other than the web server where the deceptive route is not returned to the adversary.

C. Deceptive Route

Our Python analysis script finds that the the most vulnerable node in the example topology is R3 with C_B=0.714.

Listing 4: Traceroute results (after deception). Device labels from Figure 3 added for reference.

```
traceroute to 192.168.5.2 (192.168.5.2),
30 hops max, 60 byte packets
 1  192.168.9.1   2.478 ms (R1)
 2  192.168.0.2  15.078 ms (Intelligent Router)
 3  192.168.4.2  22.520 ms (R5)
 4  192.168.5.2  32.739 ms (Web Server)
```

To minimize the maximum centrality, we find that adding a link between the intelligent router and R5 yields a new betweenness value for R3 of C_B=0.381. The resulting deception is generated as the kernel configuration of Figure 1. In the configuration file, 192.168.0.2 corresponds to the intelligent router, 192.168.4.2 to R5, and 192.168.5.2 to the destination web server. Figure 3 illustrates the deceptive route described by the configuration file. The solid arcs represent the truthful route provided to the adversary in response to a traceroute probe. Once the probe reaches the intelligent router, the adversary is then deceived. The dashed arcs represent the deceptive route supplied to the adversary by the intelligent router.

Listing 4 shows the actual output of traceroute when run by the adversary when the deception implementation is deployed on the testbed. From Listing 4 we see that the returned deception matches the configuration file. Because R1 is in front of the intelligent router, it truthfully reports its presence to traceroute. The intelligent router also truthfully reports its presence to the adversary, but all subsequent packets sent elicit deceptive, spoofed replies generated by the intelligent router in response. Finally, the adversary's traceroute probe receives the Port Unreachable message it expects for the destination 192.168.5.2 and terminates normally.

V. Conclusion

We implement a topology deception kernel module for a Linux router. The module inspects traffic at network ingress and identifies inbound UDP traceroute probes to which it generates spoofed replies to provide the illusion of any topology of the deceiver's choosing.

Our research demonstrates that a UDP-based traceroute probe may be deceived in non-trivial ways. However this initial effort leaves several avenues for future work [21]. For example, we considered other deceptive implementations. In a centralized approach, the intelligent router might identify incoming traceroute probes and shunt them into an enclave of virtual machines where a network is simulated to provide the deceptive topology. In a more distributed approach, rather than shunting traffic, the probe is permitted to traverse the true network. However the true network contains a set of virtual machines that are integrated into the topology to provide deceptive nodes and links. We leave a complete exploration of the benefits of each alternate approach to future work.

However, an implementation targeting UDP-based traceroute alone is insufficient. An adversary can easily run traceroute-style probes using TCP or ICMP rather than UDP. Future work should consider the fidelity of deception against a variety of sophisticated mapping techniques. Furthermore, a topological deception is itself not the goal. As is the case with operations in the physical domain, military deception in cyberspace should be employed in support of other operations. For instance, topological deception may be used to augment the capabilities of a honeypot to lure an attacker into revealing his or her tactics.

As the cyber domain continues to grow in importance to military operations, novel techniques are required to defend networks against increasingly sophisticated attackers. Our work takes inspiration from recent moving target defense initiatives [10] that attempt to increase the complexity and cost of an adversary's attack by shifting and changing over time. Although the implementation of topological deception in this work represents proof-of-concept only, ongoing development of the research presented herein may well complement other cyber operations and prove to be a useful application of military deception and moving target defense in cyberspace.

References

[1] U.S. Department of Defense. (2011, July) Department of Defense Strategy for Operating in Cyberspace.

[2] D. Butskoy. (2013) traceroute. [Online]. Available: http://traceroute.sourceforge.net/

[3] *Internet Control Message Protocol*, Internet Engineering Task Force Std. 792, 1981. [Online]. Available: http://www.ietf.org/rfc/rfc792.txt

[4] C. Trowbridge, "An overview of remote operating system fingerprinting," SANS Institute, Tech. Rep., July 2003.

[5] L. Spitzner, *Honeypots: Tracking Hackers*. Boston, MA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[6] B. Whaley, "Toward a general theory of deception," *Journal of Strategic Studies*, vol. 5, no. 1, pp. 178–192, March 1982.

[7] R. Beverly, "A robust classifier for passive TCP/IP fingerprinting," in *Passive and Active Network Measurement*, 2004, pp. 158–167.

[8] M. Smart, G. R. Malan, and F. Jahanian, "Defeating TCP/IP stack fingerprinting," in *Proceedings of the 9th conference on USENIX Security Symposium - Volume 9*, 2000, pp. 17–17.

[9] E. E. Frederick, "Testing a low-interaction honeypot against live cyber attackers," M.S. thesis, Naval Postgraduate School, Monterey, CA, 2011.

[10] S. Jajodia, *Moving target defense: creating asymmetric uncertainty for cyber threats*. Springer Science+ Business Media, 2011, vol. 54.

[11] J. Yuill, D. Denning, and F. Feer, "Using deception to hide things from hackers: Processes, principles, and techniques," *Journal of Information Warfare*, pp. 26–40, 2006.

[12] B. Augustin, X. Cuvellier, B. Orgogozo, F. Viger, T. Friedman, M. Latapy, C. Magnien, and R. Teixeira, "Avoiding traceroute anomalies with paris traceroute," in *Proceedings of the 6th ACM Conference on Internet measurement*, 2006.

[13] T. Liston. (2013) LaBrea. [Online]. Available: http://labrea.sourceforge.net/

[14] L. C. Freeman, "A Set of Measures of Centrality Based on Betweenness," *Sociometry*, vol. 40, no. 1, pp. 35–41, Mar. 1977.

[15] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, Jun. 1998.

[16] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11 – 15.

[17] IANA, "IP Option Numbers," 2013. [Online]. Available: http://www.iana.org/assignments/ip-parameters/ip-parameters.xml

[18] The Netfilter.org Project. (2013) iptables. [Online]. Available: http://www.netfilter.org/

[19] D. Malone and M. J. Luckie, "Analysis of ICMP Quotations," in *Passive and Active Network Measurement*, 2007, pp. 228–232.

[20] J. Grossman, B. Marsili, C. Goudjil, and A. Eromenko. (2013) GNS3 Graphical Network Simulator. [Online]. Available: http://www.gns3.net/

[21] S. T. Trassare, "A technique for presenting a deceptive dynamic network topology," M.S. thesis, Naval Postgraduate School, Monterey, CA, 2013.