REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
31-08-2011	Final Report	01-09-2010 - 31-08-2011

4. TITLE AND SUBTITLE
Mapping Attack Paths in Black-Box Networks Through Passive Vulnerability Inference

5a. CONTRACT NUMBER

5b. GRANT NUMBER
FA9550-08-1-0157

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Anup Ghosh, Steve Noel, Sushil Jajodia

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
GMU Center for Secure Information Systems
4400 University Dr, MS 5B5
Fairfax, VA 22030

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Office of Scientific Research
875 N. Randolph St.
Suite 325
Arlington, VA 22203

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
AFRL-OSR-VA-TR-2012-0243

12. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution Statement A -- Approved for Public Release

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This project investigates stealthy techniques for mapping attack graphs through black- box networks.
This provides a powerful new capability for network reconnaissance and attack planning, when open scanning is not an option.
We employ purely passive inference, as well as new hybrid passive/active techniques that provide more comprehensive attack plans while maintaining nearly zero risk of detection.
We infer network configuration (topology, devices, services, etc.), as well as functional semantics of network components for intelligent targeting. We map discovered network elements to potentially exploitable vulnerabilities.

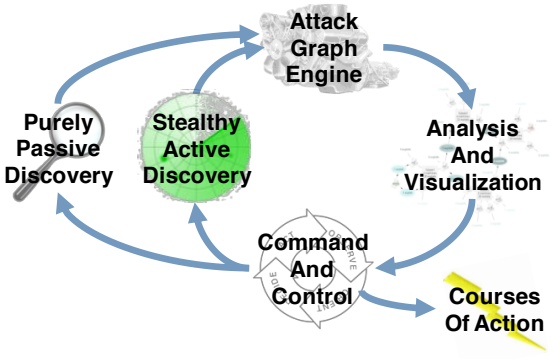
15. SUBJECT TERMS
Security. Passive vulnerability inference.

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Robert Herklotz
UNCLAS	UNCLAS	UNCLAS		35	19b. TELEPHONE NUMBER *(Include area code)* 703-696-6565

The Center for Secure Information Systems
Research Building I
George Mason University
Fairfax, VA 22030-4444

FINAL REPORT

MAPPING ATTACK PATHS IN BLACK-BOX NETWORKS THROUGH PASSIVE VULNERABILITY INFERENCE



BY

CENTER FOR SECURE INFORMATION SYSTEMS

AUGUST 30, 2011

CONTENTS

FIGURES

1. INTRODUCTION

This project investigates stealthy techniques for mapping attack graphs through black-box networks. This provides a powerful new capability for network reconnaissance and attack planning, when open scanning is not an option. We employ purely passive inference, as well as new hybrid passive/active techniques that provide more comprehensive attack plans while maintaining nearly zero risk of detection.

We infer network configuration (topology, devices, services, etc.), as well as functional semantics of network components for intelligent targeting. We map discovered network elements to potentially exploitable vulnerabilities. From this we find the possible multi-step vulnerability paths through a target network, and recommend best courses of action (e.g., additional information to gather, new hosts to control). We demonstrate purely passive inference of topology, protocols, and services of target networks, as well as a novel approach for stealthy scanning through firewalls via collaboration with an outside agent.

Research contributions of this project include the following:

- Methods of passive inference from network traffic:
  - OS detection via web browser user agent messages.
  - Detection of network services and their possible vulnerabilities.
  - Patterns of communication among workstations and servers.
  - IP packet time-to-live (TTL) for inferring network topology.
- "Inside-Out Collaborative Scanning" for stealthy network scanning:
  - Exploits inherent design weakness of current-generation firewalls.
  - Leverages insider presence (e.g., software agent) to create hole through firewall.
  - Allows outside collaborator to access (scan or exploit) arbitrary target machines inside network.
  - Insider sends only single packet to open a target port, spoofing the outside collaborative scanner address.
  - Insider never reveals own IP address.
  - Technique experimentally verified.
- Overall system architecture for stealthy passive/active network reconnaissance tool.
- Automatic discovery and change detection for Layer-2 devices (switches).
- Capabilities for visual analysis and reasoning for network reconnaissance.
- Mapping from discovered network elements (target subnets, hosts, services, connectivity, etc.) to input model for multi-step attack graph modeling for intelligent attack planning.

2. OVERVIEW OF APPROACH

Figure 1 is an overview of our approach to stealthy inference and attack planning for black box networks. In this approach, we infer as much as possible about a target network through purely passive observation. We then use what is learned passively to guide stealthy active scanning techniques. Network reconnaissance data, from both passive and active modes, are used as inputs for attack graph modeling. Analysis of the resulting network model shows the possible multi-step paths through the target network, organized as a concise attack graph.

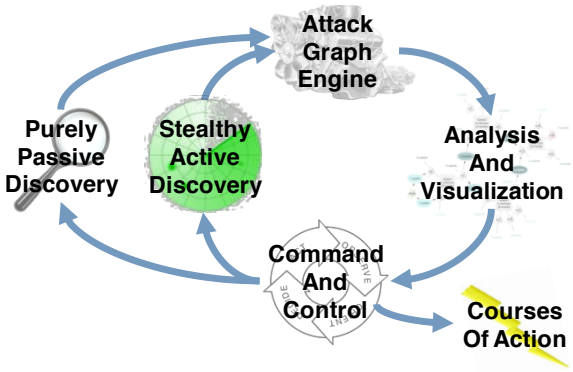


Figure 1: Components of stealthy black-box network attack planning.

Through visual analysis and interactive navigation of the attack graph, analysts can plan optimal strategies for attack against the target network. This may include steps for gathering more network reconnaissance data, or exerting greater control over the network. Our approach is agent-based, in that software agents either inside the target network or cooperating agents outside the target network are used to gather and communicate reconnaissance data. In some circumstances, the vantage points of inside agents may allow for penetration deeper into the target network, through deployment of new agents.

In the simplest possible deployment architecture, shown in Figure 2(a), an inside agent is positioned inside a target network. Such an inside agent could determine its own local host configuration, and could passively observe traffic within its network segment and infer some information about other hosts. For example, it could detect which outside hosts are allowed to connect to target hosts, thus inferring firewall holes. It could also detect services on target hosts, and in some cases operating systems and known vulnerabilities. The inside agent could then communicate its reconnaissance data to an attack planning component outside the target network, e.g., via some covert channel. Although the details of such covert communication are outside the scope of this project, we note that firewalls and intrusion detection systems generally consider such outbound traffic benign. Also, the inside agent could communicate to the outside by spoofing the address of another inside host, never revealing its own address.

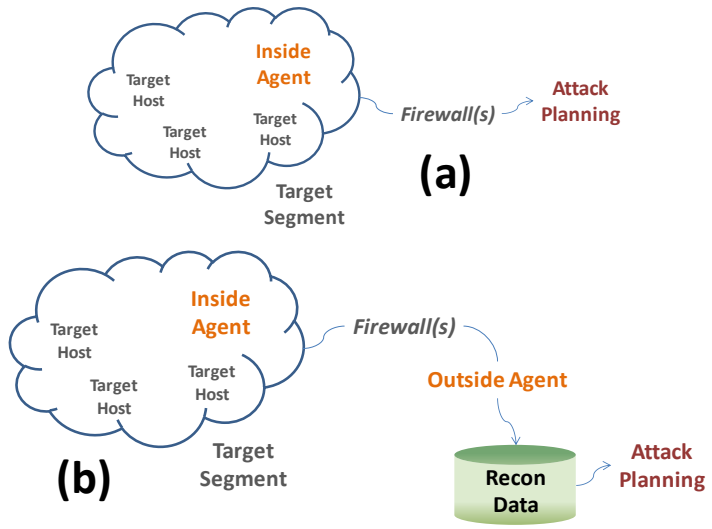


Figure 2: Simple deployment architectures.

As we describe in Section 4, we introduce new techniques for stealthy network reconnaissance that involve cooperation between the inside agent and an outside agent. This is depicted in Figure 2(b). The inside agent sends minimal traffic to the outside agent, by spoofing the source address of an inside target host. This opens a temporary hole in the firewall that allows the outside agent to connect to the target host. Under this architecture, the outside agent must share its reconnaissance data with an attack planning component. Ideally, this is done through an intermediary outside service that interfaces with the outside agent and stores its reconnaissance data. That same service can then interact with the attack planning component to populate its predictive attack models.

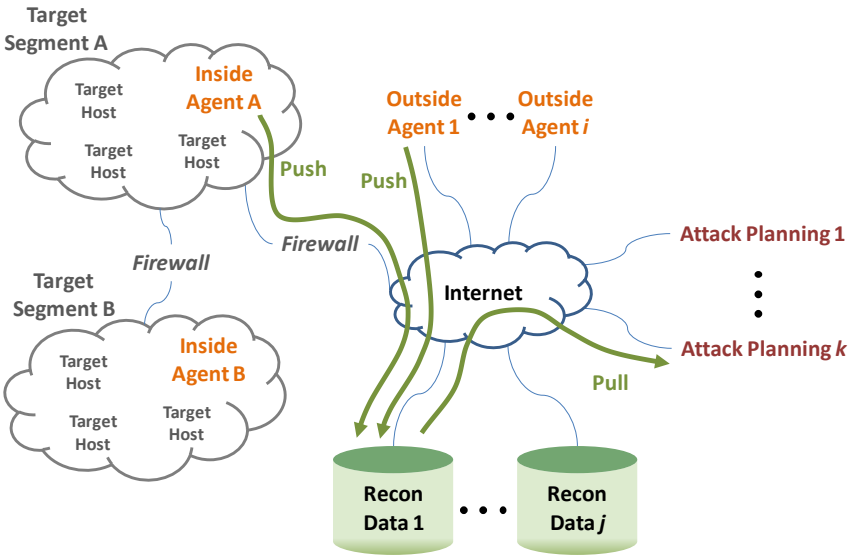


Figure 3: Distributed deployment architecture.

More realistic deployment architecture is depicted in Figure 3. This shows inside agents deployed on two different segments of the target network. For example, Inside

Agent A could have injected executable code into traffic destined for Target Segment B as it traversed its local Target Segment A, thus compromising a host in Segment B. This architecture is more distributed than Figure 2, with both inside agents and outside agents communicating with one or more outside reconnaissance data servers. This helps minimize detection by distributing over multiple destination addresses, as well as provides redundancy in the case of server or network failures. Similarly, attack planning workstations could communicate with one or more reconnaissance data servers.

3. PASSIVE NETWORK INFERRENCE

A variety of reconnaissance data about a network may be gleaned passively. This includes internal hosts in the network, as well as external hosts that communicate with them. Host names, internet (IP) addresses, and hardware (MAC) addresses are readily detected for internal hosts, for the local segment of an inside agent. To some degree of accuracy, host operating system and version can be fingerprinted, as well as client applications and services. A machine's logical location (number of network hops) can also be inferred with some confidence. Through analysis of the application layer, usernames and passwords can sometimes be discovered, as well as sensitive data. Indeed, some network components are discovered passively that are *not* discovered through active scanning [1][2][3][4][5], such as some client-side applications and their vulnerabilities, port-based trust relationships, or services that are not available during the time of the active scan. Also, active scans are snapshots at a particular point in time, while passive scanning can work constantly.

Currently available tools for passive network scanning include Tenable PVS (vulnerability scanner) [6], p0f [7] and ettercap [8] (OS fingerprinting), and Snort IDS [9], which can be configured for general-purpose traffic analysis. Also, some commercial tools employ passive detection of hosts on the network, and then target those hosts with active scans.

However, passive scanning does have significant limitations. It cannot detect the patch level of a host OS, which is important for mapping to known vulnerabilities. Services that are available but not communicating with other hosts are not discovered. Moreover, in many cases detection accuracy is low. In one study [10], a high error rate is reported for ettercap for passive OS fingerprinting. Our own informal tests with the p0f tool showed significant OS misclassifications. Also, while techniques exist for detecting services and their vulnerabilities, this largely relies on service banners, which can be easily changed by administrators.

Because of the inherent limitations of passive network inference, we supplement purely passive techniques with stealthy active techniques. We passively detect network hosts and their services, and then rely on cooperation with an outside agent for active scanning that detects vulnerabilities in those services. We can also leverage the vantage point of an insider presence for injection attacks that deploy insider agents on other hosts, e.g., deeper in the network. These hybrid passive/active techniques are described further in Section 4.

3.1 OS and Application Detection

As an example of detecting a host OS passively, consider Figure 4. This shows instances of Microsoft Windows Browser Protocol [11] (used for host name resolution), which we collected from live network traffic. This protocol reports major and minor OS version. This allows us to infer the OS to some degree of accuracy, based on known values for the various Windows platforms [12], as shown in Figure 5.

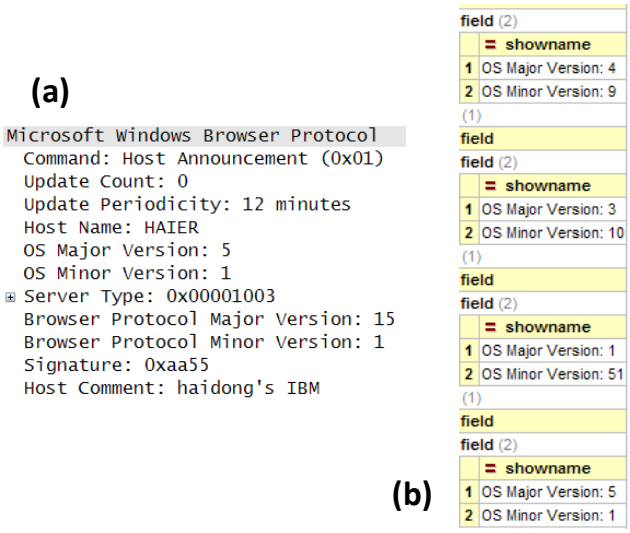


Figure 4: MS Browser Protocol for OS detection.

For example, the observed IP packet in Figure 4(a) shows Major 5/Minor 1. From Figure 5, we infer that this host is running Windows XP or XP Service Pack 2. From other detected packets shown in Figure 4(b), hosts with Major 4/Minor 9 and Major 3/Minor 10 are older than Windows 98.

	Win 95	Win 95 OSR2	Win 98	Win SE	Win Me	Win NT 4	Win 2000	Win XP	Win XP SP2	Win 2003 Server	Win Vista	Win Vista SP1
PlatformID	1	1	1	1	1	2	2	2	2	2	2	2
Major Version	4	4	4	4	4	4	5	5	5	5	6	6
Minor Version	0	0	10	10	90	0	0	1	1	2	0	0
Build	950*	1111	1998	2222	3000	1381	2195	2600	2600	3790	6000	6001

Figure 5: Windows Major and Minor versions (from [12]).

As another example, we employ passive inference to detect applications running on a host. In this case, we detect web browsers being used on a host. As shown in Figure 6, we passively observe HTTP User-Agent headers on the network. Web clients use the User-Agent to report to web servers the kind of browser they are. In this example, we install 3 different web browsers on a host – Internet Explorer, Firefox, and Safari. We then use each browser to connect to a web server, and passively observe the web traffic. In Figure 6(a), Figure 6(b), and Figure 6(c), we show the observed User-Agent headers for Internet Explorer, Firefox, and Safari (respectively), along with the actual full version of each browser. For Firefox and Safari, the User-Agent matches the actual 3-level

versions exactly. For Internet Explorer, it correctly identifies to Version 7.0 (2 version levels). An on-line database is available [13] giving User-Agent headers for a variety of web applications, including browsers, search engines, and web crawlers.

Figure 6: User-Agent headers for web browser identification.

3.2 Service Detection

Other than host OS and client applications, we can also use passive inference to map services on a target network. A basic function is the analysis of protocols observed in network traffic. Figure 7 shows network traffic broken down by packets. Each packet contains a number of protocols in the network stack. For example, the first packet in Figure 7 is for address resolution (ARP), i.e., mapping from IP address to MAC address. Other packets might be for domain name resolution (DNS), mapping domain names to IP address, web (HTTP) packets, etc. These higher-level protocols are encapsulated above the lower-level frames and Ethernet protocol.

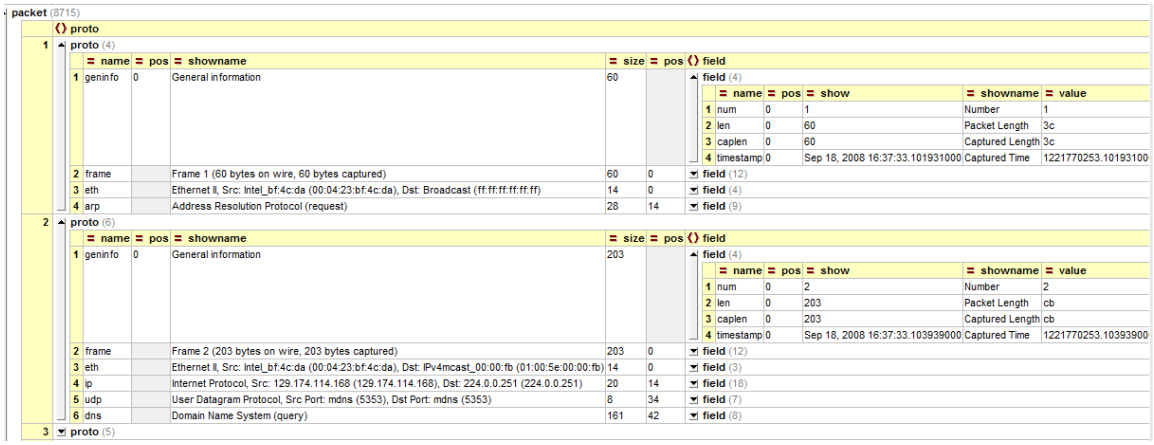


Figure 7: Structure of observed traffic packets.

As an initial analysis, we show all observed packets, plotted by source and destination (Figure 8). This represents 8,715 packets observed on a live network over the course of

about 15 minutes. From this plot, we can infer the likely existence of 2 servers (columns), which are each communicating with multiple clients (rows).

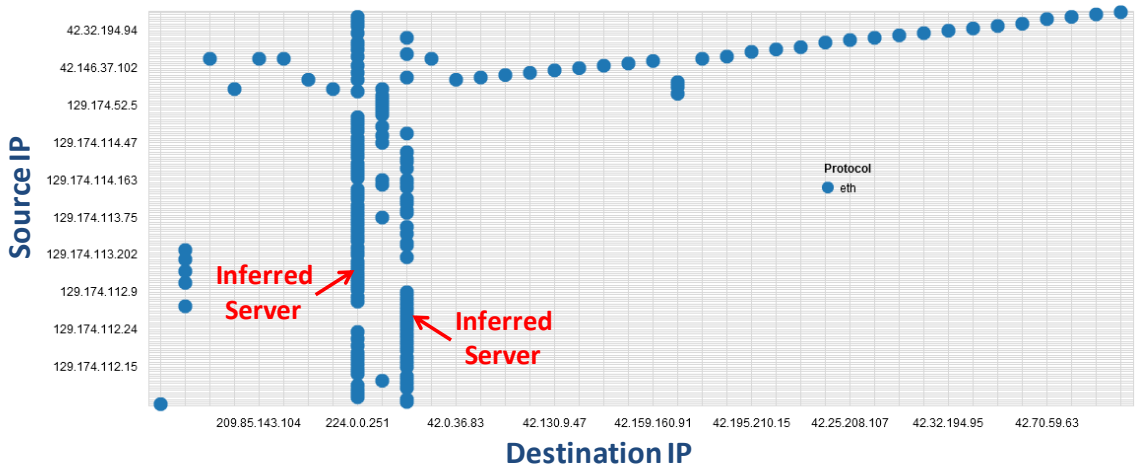


Figure 8: Inferring two network servers.

To further classify these servers, we go higher up the network stack and plot source and destination for traffic specific to DNS (Figure 9). This clearly shows that we can infer the IP address of a DNS server (column) in this network. Domain name resolution is a critical network service, so that this DNS server becomes an important potential attack target.

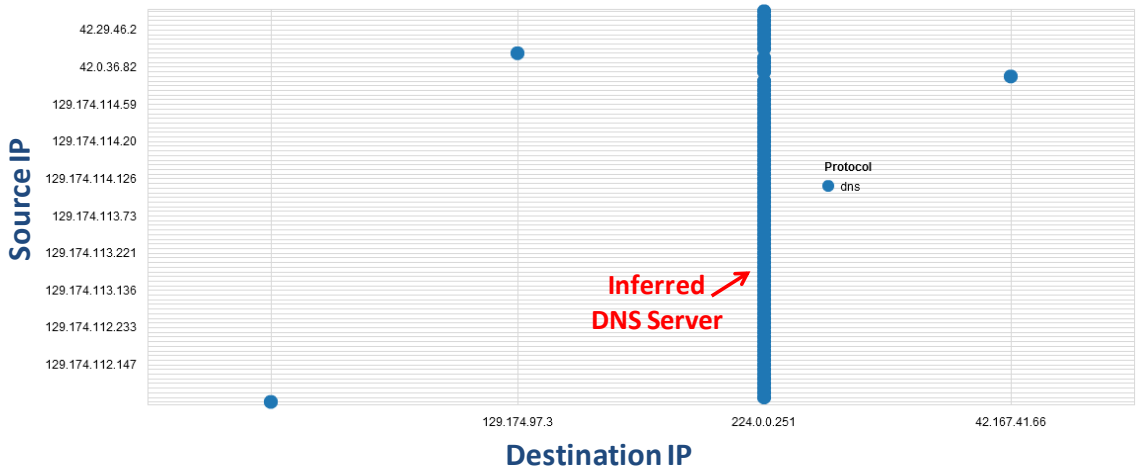


Figure 9: Inferring a DNS server.

For this same traffic, we wish to understand which protocols are present beyond the routine network management protocols. The idea is to get a better understanding of some of the application-layer traffic. To do this, we exclude certain protocols from the analysis, as shown in Figure 10.

```
<xsl:for-each select="pdml/packet">
    <xsl:for-each select="proto">
        <xsl:if test="not(
                @name = 'ip' or
                @name = 'tcp' or
                @name = 'zip' or
                @name = 'dhcpv6' or
                @name = 'ddp' or
                @name = 'aarp' or
                @name = 'nbp' or
                @name = 'redbackli' or
                @name = 'geninfo' or
                @name = 'frame' or
                @name = 'eth' or
                @name = 'arp' or
                @name = 'udp'  or
                @name = 'fake-field-wrapper' or
                @name = 'bootp' or
                @name = 'stp' or
                @name = 'llc' or
                @name = 'cdp' or
                @name = 'dns' or
                @name = 'cups' or
                @name = 'ipv6' or
                @name = 'icmpv6' or
                @name = 'sonmp'
                )">
```

Figure 10: Set of excluded protocols.

The resulting plot of source and destination addresses is in Figure 11. From this, we infer the presence of a web server. We also see isolated user of encrypted traffic (secure socket layer) and multicasting (e.g., for gaming or video streaming). This kind of interesting traffic should be analyzed further on a target network. We also infer the existence of a Windows network, including possible file sharing that could be vulnerable to attack.

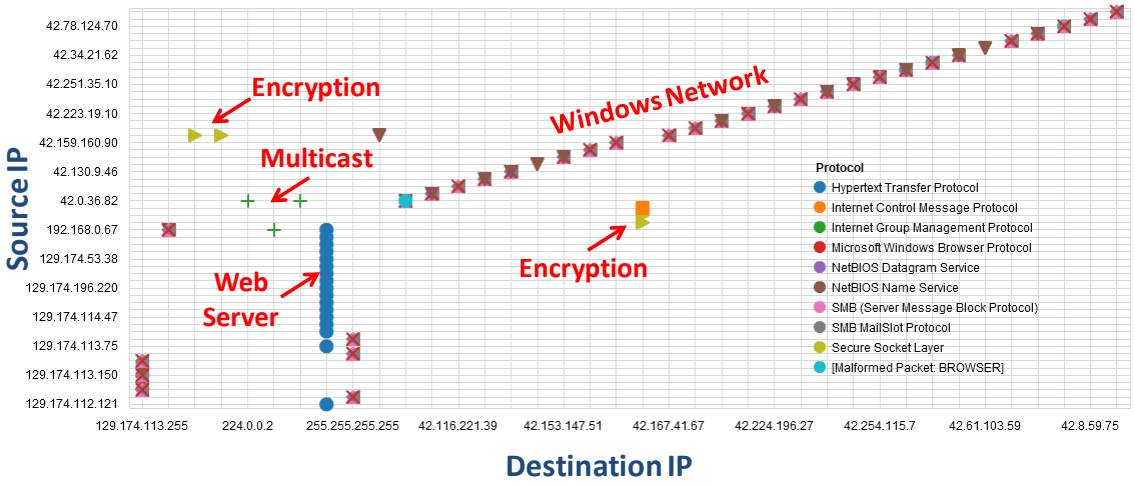


Figure 11: Interesting inferred network activity.

To make further inferences about passively observed traffic, we wish to go beyond protocol analysis, to determine exact network services and their versions. This information would help us map service versions to known vulnerabilities. One approach is to search network traffic for banners provided by services to their clients, which inform clients about service configuration. But to do this requires a database of patterns that

match known service banners. We can then passively observe when clients make their normal connections to a service, and match to our known pattern for that service banner.

As an experiment for building a database of known service banners, we mine data from the Nessus vulnerability scanner [14]. In particular, we search through the set of Nessus scripts for active detection of vulnerabilities. In the case of service banners, Nessus acts as a client, connecting to a target service, then analyzes the resulting banner for known patterns. We analyze the Nessus scripts for a signature of such banner pattern matching. With this method, we extracted almost 400 patterns for known service banners. With this information, we can map directly to known vulnerabilities for those services. Figure 12 shows examples of such service banner matching patterns.

```
3com_3cserver_ftp_overflow.nasl:if ( egrep(pattern:"^220 3Com FTP Server Version 1\.[01]([^0-9]|\.)", string:ftpbanner) ||
3com_3cserver_ftp_overflow.nasl:     egrep(pattern:"^220 3Com 3CDaemon FTP Server Version [0-2]\.", string:ftpbanner))
4d_webstar_ftp_overflow.nasl:     egrep(pattern:"^Server: 4D_WebSTAR.*/([0-4]\.|5\.([0-2]\.|3\.[0-2][^0-9]))", string:banner) )
4d_webstar_ftp_overflow.nasl: if (egrep(string:ftpbanner, pattern:"^220 FTP server ready\."))
4d_webstar_information_disclosure.nasl:     egrep(pattern:"^Server: 4D_WebSTAR.*/([0-4]\.|5\.([0-2]\.|3\.[0-2][^0-9]))", string:banner) )
4d_webstar_remote_buff_overflow.nasl:     egrep(pattern:"^Server: 4D_WebSTAR.*/([0-4]\.|5\.([0-2]\.|3\.|4[^.]))", string:banner) ) securit
4d_webstar_symb_link.nasl:     egrep(pattern:"^Server: 4D_WebSTAR.*/([0-4]\.|5\.([0-2]\.|3\.[0-2][^0-9]))", string:banner) )
4d_webstar_symb_link.nasl: if ( egrep(string:ftpbanner, pattern:"^220 FTP server ready\."))
Black_JumboDog_FTP_overflow.nasl:       if ( egrep(pattern:"^220 .*BlackJumboDog.* Version 3\.([0-5]\.[0-9]+|6\.[01]([^0-9]|$))", string:
DDI_IPSwitch-IMail-SMTP-Buffer-Overflow.nasl:              egrep(pattern:"IMail 6\.0[1-6] ", string:banner)      ||
DDI_IPSwitch-IMail-SMTP-Buffer-Overflow.nasl:              egrep(pattern:"IMail 6\.0 ", string:banner)           ||
DDI_IPSwitch-IMail-SMTP-Buffer-Overflow.nasl:              egrep(pattern:"IMail [1-5]\.", string:banner)
DDI_Netware_Management_Portal.nasl:if (egrep(pattern:"^Server: NetWare Server", string:banner) ) security_hole(port);
DDI_ws_ftp-server-cpwd-bo.nasl: if(egrep(pattern:".*WS_FTP Server (((1|2)\..*)|(3\.((0(\..*){0,1})|(1\.1))))", string:banner))
OmniHTTPd_pro_post_dos.nasl:if ( ! egrep(pattern:"^Server: OmniHTTPd", string:banner ) ) exit(0);
RA_www_detect.nasl: if (egrep(pattern:"^Server: *RemotelyAnywhere", string:banner))
SHN_Sendmail_DoublePipe.nasl: if(egrep(pattern:"Sendmail.*(8\.8\.[89]|8\.9\..*|8\.1[01]\..*|8\.12\.[0-7][^0-9])/", string:banner))
ability_ftp_overflow.nasl:if ( egrep(pattern:"^220 Welcome to Code-Crafters - Ability Server ([0-1]\..*|2\.([0-2]|3[0-4]))[^0-9]", string:
abyss_overflow.nasl:if(egrep(pattern:"^Server: Abyss/(0\..*|1\.(0\..*|1\.[0-5])) ", string:banner))
aix_ftpd.nasl:if ( ! egrep(pattern:".*FTP server .Version 4\.", string:banner) ) exit(0);
alibaba_overflow.nasl:if(!egrep(pattern:"^Server:.*[aA]libaba.*", string:banner)) exit(0);
```

Figure 12: Sample patterns for detecting known network services.

3.3 Network Communication Patterns

Beyond simply enumerating hosts, operating systems, applications, and services on a target network, it is important to consider more complex interaction among these components. For this, we employ visualization techniques to make such complex patterns easier to understand. A basic kind of visual analysis is to show patterns of communication across the network, from traffic observed by an inside agent. Because network traffic is so voluminous, it is important to do this kind of analysis at the right levels of abstraction for informed decision making.

For example, consider Figure 13(a). This shows patterns of network traffic at the packet level, which is too detailed for easy understanding. In contrast, Figure 13(b) shows the same network traffic, this time aggregated to host-to-host communication. The patterns of communication are much clearer.

Thus aggregation serves to help manage complexity for larger network pattern graphs. A complementary approach is to divide a pattern graphs into its connected components, in this case, set of intercommunicating hosts. This is shown in Figure 14. Figure 14(a) shows host-to-host communication patterns for a given set of observed traffic. Figure 14(b) shows the same patterns, but this time separated by connected components. In this case we automatically detected 6 different connected components, and plotted them separately. In an operational attack planning system, each component could be analyzed independently, focusing on a single set of intercommunicating hosts.

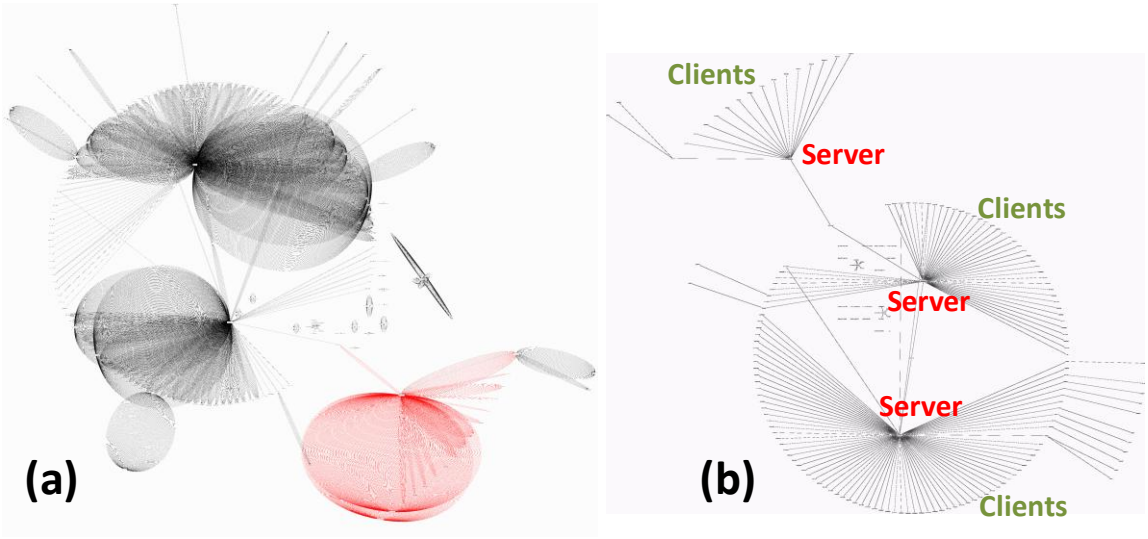


Figure 13: Levels of abstraction for network communication patterns.

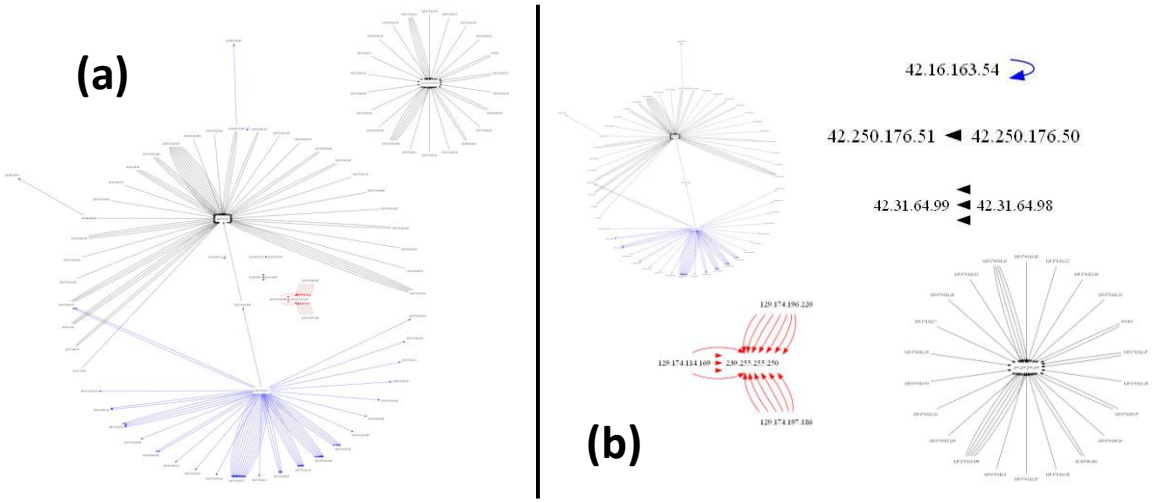


Figure 14: Separating traffic patterns into independent components.

3.4 Local Segment Detection

In our approach, attack graphs provide a powerful methodology for network attack planning. A critical requirement for attack graph analysis is to know the different segments within a target network. In our attack graph model, an important abstraction is the protection domain, which represents set of machines with full connectivity to one another's vulnerable network services, e.g., not filtered by firewalls.

One approach is to infer the local segment of an inside agent host using the time to live (TTL) field of IP packets. TTL is a limit on the number of network hops that a packet can make before it is discarded. For each hop, the TTL value is decremented by

one. By exploiting the fact that most operating systems use known initial values of TTL (powers of 2), we can infer the number of hops a packets has made at the point we observe it. As an experiment, we analyze observed TTL values to infer which host are in the inside agent's local segment. If a host is observed as sending a packet with a known initial value, we infer that it is the initial source of this packet, and the host is thus in our local segment. All other observed hosts are then inferred as being in different segments.

Analysis of TTL for predicting network topology is not always accurate, particularly when using only passive techniques. For example, packets crafted by custom applications (versus the operating system network stack) may have arbitrary initial values. In our experiments, our heuristic correctly detects all hosts in the local segment if they *send* traffic. But for hosts in the local segment that *receive* traffic only, and do not send it, our heuristic incorrectly classifies them as outside the local segment. However, once such a host actually sends a packet with a known initial TTL value, we correctly classify it.

Once we infer network hosts and classify them as inside or outside the local segment, we build a corresponding network model as input to attack graph analysis. To generate such a network model, an inside agent observed 3,833 packets over a period of 11 minutes on a live network. We then applied TTL analysis to the observed traffic, inferring a set of 119 hosts on the agent's local segment. The remaining 47 observed hosts were inferred as communicating from outside the local segment. For each outside host, if we observed communication with an inside host, we modeled that as a connection to a potentially vulnerable client or service on the inside host. The resulting network model is shown in Figure 15.

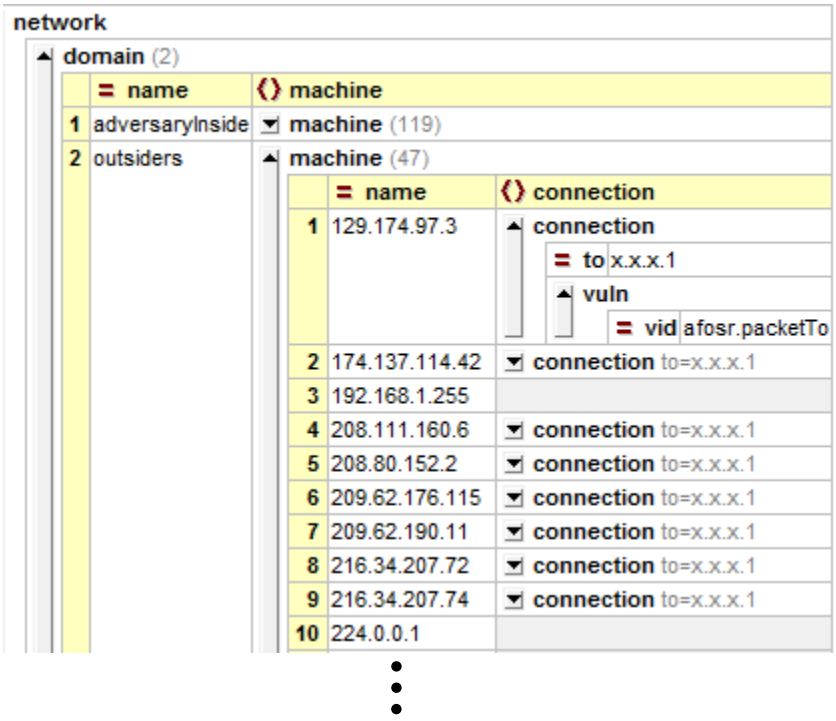


Figure 15: Inferred network model for input to attack graph analysis.

This network model is a first approximation for attack graph analysis. To refine this model, we need to infer any known vulnerabilities with client or server software on the target hosts, e.g., though the techniques described in Sections 3.1 and 3.2. Still, the network model in Figure 15 serves as a good initial input to an attack graph analysis, which we describe in Section 7.

4. STEALTHY ACTIVE TECHNIQUES

In general, stealthy exploitation of a target machine can be challenging. In a perfect scenario, one would like to know extensive details about the target, including operating system version, patches, configurations, language packs, application versions and configurations, network connectivity, host protection mechanisms, and sometimes even the number of host CPUs [15].

Much of this target network information is simply not available through passive network observation. Active vulnerability assessment tools such as Nessus avoid this problem by directly probing for vulnerabilities. For example to test for a particular buffer-overflow directory traversal vulnerability, a Nessus script might attempt to actually to overflow the input buffer on a target host's service, and then traverse to an arbitrary file directory on the host. While this approach works for blue-team vulnerability assessment, it is easily detected when stealth is required.

For the first time, we introduce novel techniques that provide the accuracy of active scanning, while minimizing the risk of detection. These involve a stealthy insider in the target network, with a cooperating outsider. The advantage of this unique approach is that the outsider can generate active traffic for scanning and/or exploiting inside targets, while the insider remains stealthy. This is a key factor in maintaining stealthy inside presence in the target network. Our adversaries would consider the outsider's traffic as typical scanning attempts (e.g., via intrusion detection systems), or even ignored in cases where spoofed connections are initiated from the inside.

We introduce two fundamental types of scanning operations:

- Insider observation of existing firewall holes (Section 4.1)
- Spoofed insider to open new firewall holes (Section 4.2)

4.1 Insider Inference of Firewall Rules

In this mode of insider/outsider scanning, we infer the state of one or more firewalls in a target network. In particular, an outsider infers all firewall holes based on spoofed responses of a collaborating inside agent. In this mode, the outsider discovers all firewall holes, versus scanning a single target IP/port.

As shown in Figure 16, the insider passively observes traffic to estimate the IP address range of the target network segment. This allows the outsider to focus the probing of firewall holes. The insider passively observes which probe packets reach the inside (through the firewall), and then echoes those packets to the outsider (spoofing the other inside targets).

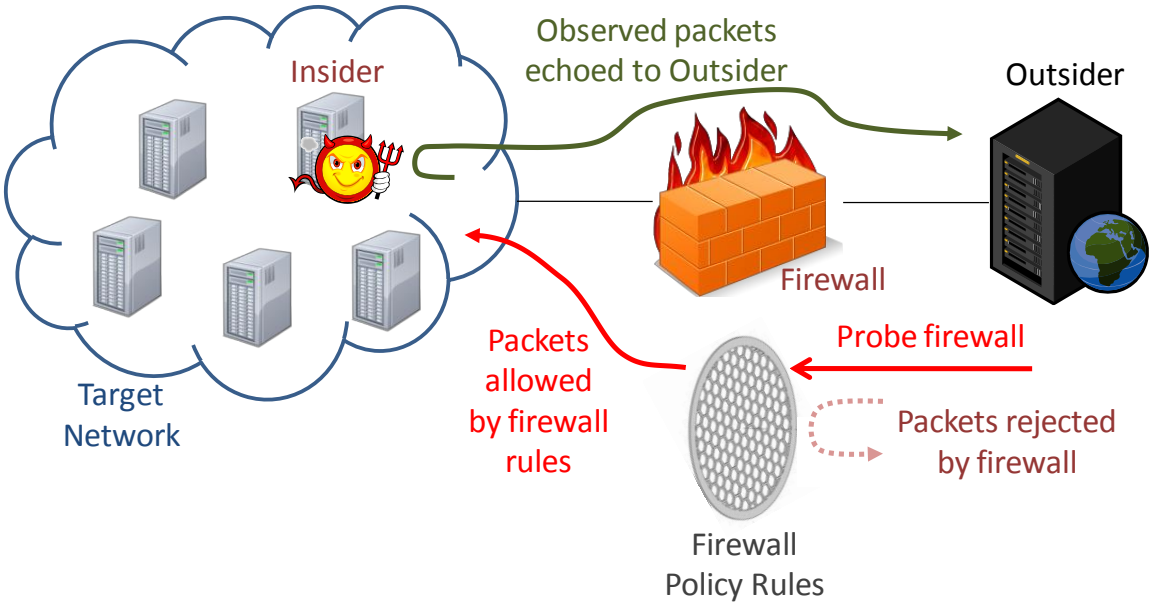


Figure 16: Stealthy inference of firewall holes in target network.

In our approach, the outsider samples the firewall rule-set space, i.e., traffic source and destination IP addresses, subnet masks, protocols (TCP, UDP, etc.) and specific ports. In this way, through feedback from the insider (spoofing the inside targets), the outsider can infer the entire set of policy rules enforced by a firewall. In the case of multiple firewalls, this approach infers the combined policy enforced by all firewall layers.

Figure 17 shows the structure of our inferred firewall policy, capturing net connectivity effects through one or more firewall layers. Because of the observational nature of our approach, our policy model is stateless, with no dependencies among rules. Thus our model is much simpler and tractable than usual firewall specification languages.

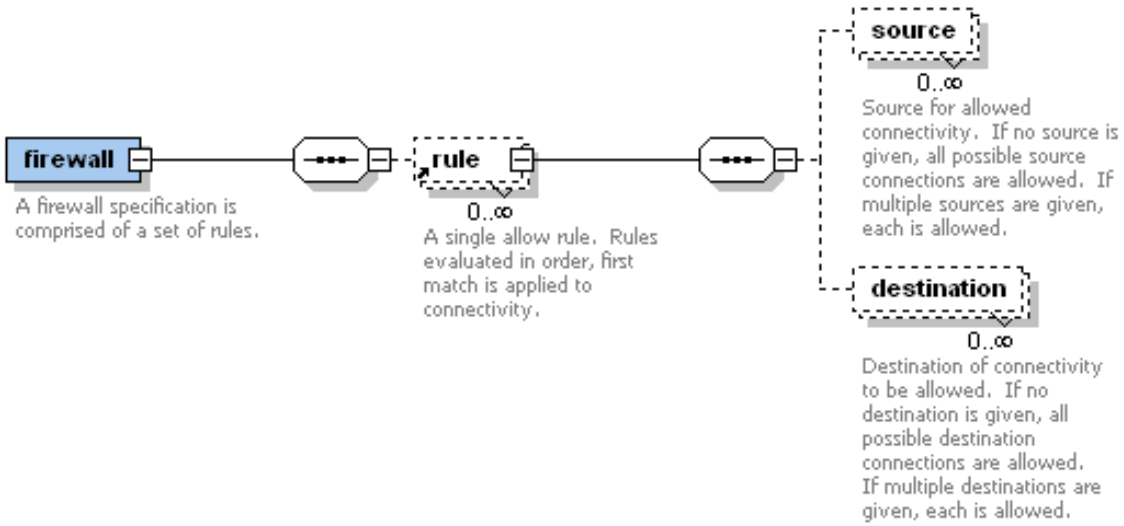


Figure 17: Structure of inferred firewall effects.

In our policy model, a firewall effects specification is a set of rules of connectivity allowed through firewall layers to inside targets. Each rule specifies connectivity from a source to destination. This corresponds to "allow" rules in usual firewall configurations. Because we are observing traffic (generated by our outsider) allowed into the network, usual "deny" rules do not apply (i.e., their effects are represented implicitly). In the configuration of operational firewall configurations, policy rules can overlap, be redundant, contradictory, etc. But in our approach we observe firewalls as black boxes, so that we can normalize any such problematic rule sets definitions.

Figure 18 shows example policy rules inferred through our approach. Each rule has a source and destination for connectivity that is observed to be allowed through the firewall(s). Sources or destinations can be single IP addresses, such as the source for the first example rule. They can also be ranges of IP addresses, according to network masks as defined in actual firewall device configurations. In the example, the second rule has such a source. Our policy model also allows the special case of "any" source and/or destination, meaning all possible IP addresses (e.g., the destination for the first example rule).

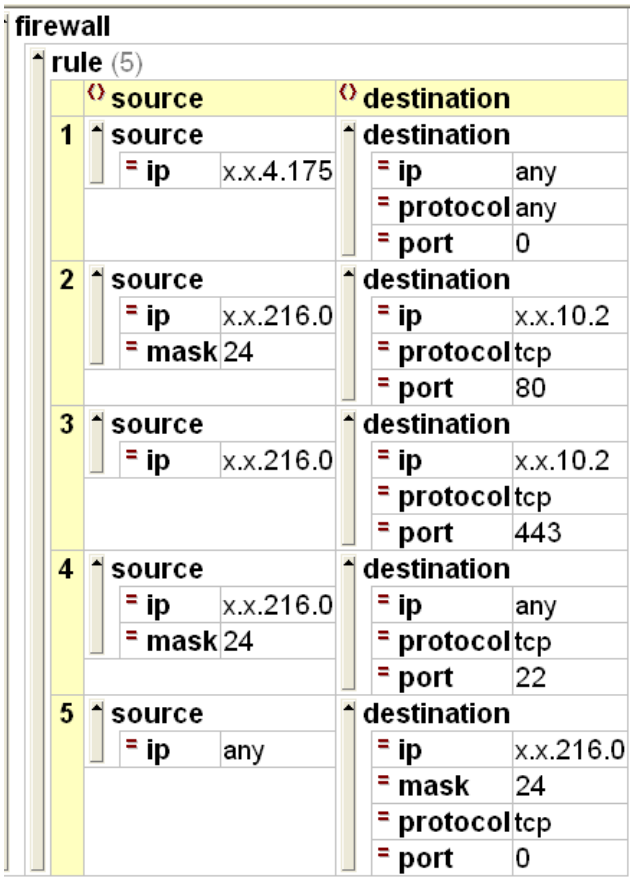


Figure 18: Example inferred firewall policy rules.

Our policy model also includes traffic protocols and destination ports for rules. This means that only the specified protocols and ports are allowed in the observed traffic. In our model, ports apply to destinations only (i.e., source ports are irrelevant), since an

attacker can set their source port as needed. Our inferred firewall policies are combined with knowledge of the vulnerabilities inferred for network targets, yielding single-step attack vectors to be combined into overall attack graphs.

In this approach to firewall inference, the stealthy insider never reveals its own address, i.e., it spoofs the addresses of other network hosts being targeted. Thus there is near zero risk of the insider being detected. This approach works through multiple layers of firewalls for inferring the combined effects of all firewall rule layers. Unlike purely passive approaches, we do not rely on opportunistic analysis of normal traffic. Instead, our cooperating outside agent probes from outside the network the insider remains undetected.

4.2 Opening Firewall Holes via Insider

In this mode, rather than only testing/observing existing firewall holes, we leverage the vantage point of the insider to actually open new firewall holes to the outsider. To the adversary employing standard intrusion detection technology, this appears as simply a normal outbound connection. But in this case, the insider is spoofing an inside target to set up the outbound connection state.

This technique exploits a flaw in the current generation firewalls, and gives the outsider unrestricted access to target hosts on the inside. The inside agent sends a SYN packet to the outsider, spoofing a target insider host.

Figure 19(a) depicts the usual situation when an outsider attempts to connect to an inside network protected by a firewall. Assuming no firewall holes (allow rules) are available from the outside, the attacker's TCP SYN packet for establishing a connection is dropped by the firewall. However, the firewall allows traffic to flow unblocked inside the network, so that inside hosts can connect to one another's services via the usual TCP handshake (SYN, SNY-ACK, ACK).

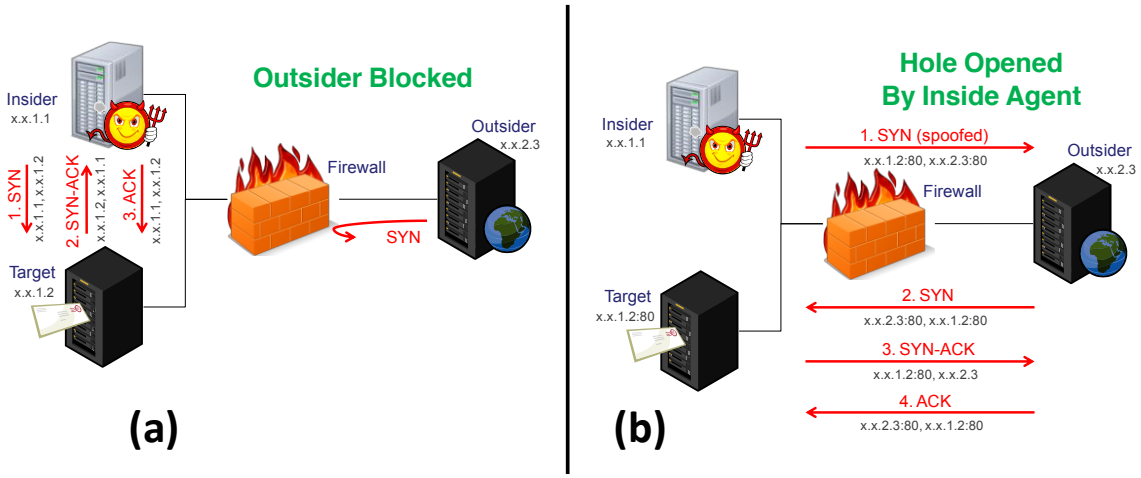


Figure 19: Inside agent opening firewall hole for outside agent.

Figure 19(b) shows our novel technique in which the insider opens a temporary hole in the firewall, allowing an outside agent to reach a target host on the inside. The insider sends a packet to the outsider, spoofing the target host as the source address. Firewalls are generally configured to allow such inside-initiated traffic. The spoofed packet sets a state in the firewall in which subsequent traffic between the source/destination IP address and TCP ports are allowed. In this case, this opens a temporary hole for a service on the target host.

Without prior communication or agreement, the insider cannot predict the TCP port that the outsider will use to connect to the target. In fact, using the usual client applications, the outside host does not know the port that will be allocated by its OS network stack. We thus attempt the connection, detect the allocated port, and then use the next highest port number as the likely next one allocated (shown in Figure 20).

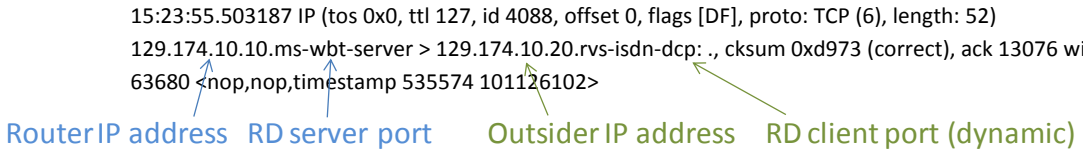


Figure 20: Outsider detects own allocated client port.

The outsider then uses the firewall hole to communicate its predicted next allocated port to the inside agent. This packet is sent to the spoofed inside target, but observed by the inside agent. The insider then spoofs a second packet, this time with the outsider's predicted next port number. A subsequent connection attempt by the outsider is then successful.

We have implemented this stealthy active approach in our laboratory network. Using a network configuration as in Figure 19, we implement the firewall function with a router configured to perform network address translation (NAT), with no port forwarding rules to allow connections from the outside. With NAT, the outsider sees only the external IP address of the router, so any attempts at scanning or otherwise compromising the inside network are unsuccessful.

The target host is running Windows XP, which by default allows connections to its remote desktop service [16], which allows another host to control its desktop remotely. Our inside agent opens a hole in the router, allowing the outside agent to connect to the target host and remotely control its desktop.

Figure 21 shows the insider spoofing the first packet to open a hole in the router, communicating the target IP and port to the outsider. It then monitors inbound traffic, looking for the response packet from the outsider. At this point, it sends a second packet, opening a hole for the outsider's remote desktop client port.

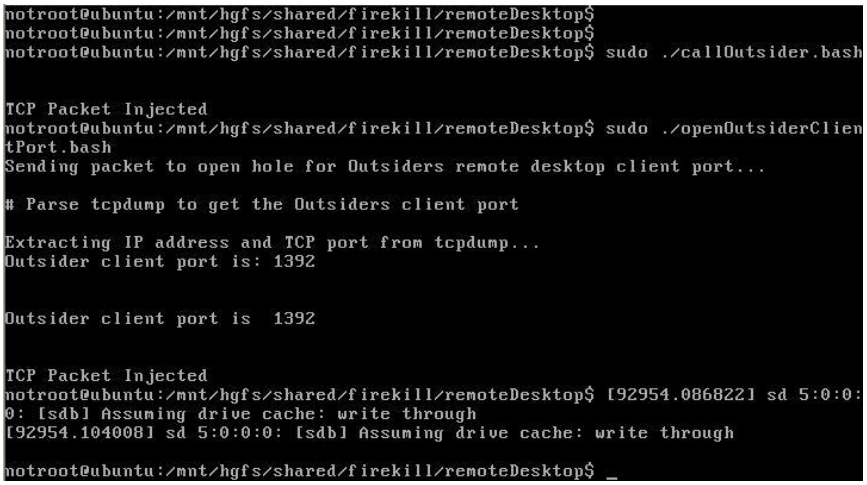


Figure 21: Insider spoofs target host packets that open firewall.

Figure 22 shows the sequence of events on the outside agent host. The outsider first receives the spoofed packet from the target network, containing the target IP address and port. It attempts to connect to that target with its remote desktop client and fails. It observes the port number allocated by its OS, and then uses the firewall hole to send that message to the inside.

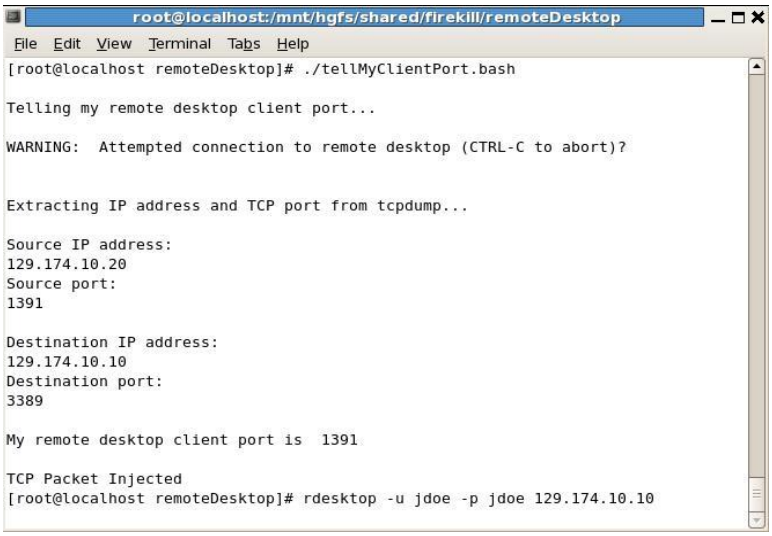


Figure 22: Outsider tells insider its next client port number.

After the insider sends a second packet with the correct client port, the outsider attempts to reconnect to the target's remote desktop service. This time the attempt is successful, and the outsider takes control of the target host, as shown in Figure 23.

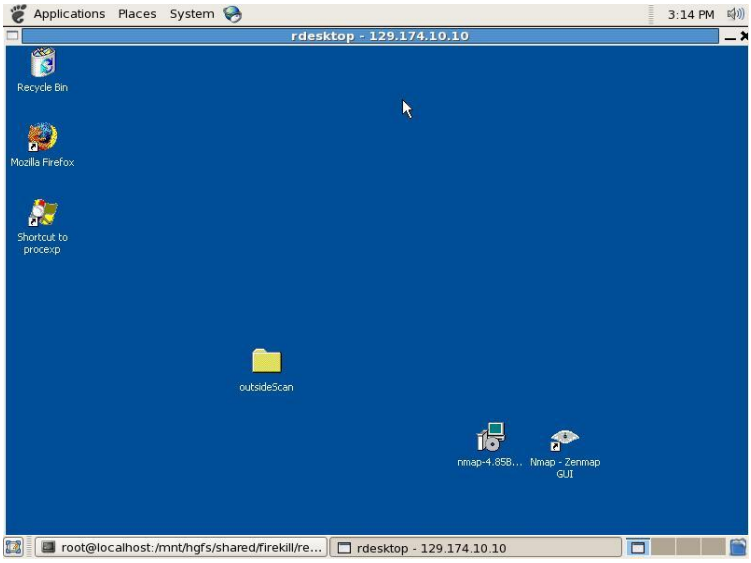


Figure 23: Outsider has taken control of inside target host.

From the firewall's point of view, this looks like the spoofed insider trying to establish a normal outbound connection. The firewall then allows subsequent traffic between the outsider and the spoofed insider, giving the outsider full access to the target (IP address and port). For this technique, the inside agent sends only single packet, and never reveals his own address.

5. DISCOVERY OF LAYER-2 TOPOLOGY

Firewalls and routers filter traffic at the Network Layer (Layer 3) [17], e.g., TCP/IP. But there is growing use of devices (switches) that provide connectivity at the Data Link Layer (Layer 2). For mapping the topology of target black-box networks, an important requirement is having accurate knowledge of Layer-2 connectivity.

We introduce an automated approach for discovering Layer-2 topology [18]. This approach leverages Layer-2 forwarding behavior. It creates packet probes that reveal and monitor underlying ethernet network topology. In some scenarios, this approach works for Layer-3 network elements as well.

Unlike previous approaches, we do not depend on network support, e.g., special protocols, management software or even software agents running on target network hosts. This approach is therefore ideal for black-box network topology discovery. We measure network physical propagation parameters including Round Trip Time (RTT) and packet loss to infer the position of switches and connected hosts. We demonstrates accurate discovery of physical network topology, with reasonable computation time even for fairly large networks.

In previous work, there have been few solutions for automatic physical topology discovery, and none of them are appropriate for black-box networks. These generally rely on very restricting deployment scenarios, such as support for Simple Network Management Protocol (SNMP) [19], vendor-specific products, or pre-installed software.

We introduce a novel approach that automates the topology discovery of the ethernet infrastructure of black-box networks. This approach explores the network from one or more vantage points using a specially crafted sequence of Address Resolution Protocol (ARP) [20] and TCP ping probes.

As shown in Figure 24, we employ a two-phase algorithm that exposes the structure of the underlying ethernet network observed from each vantage point. During the first phase, local hosts are discovered through ARP and ping broadcast requests, obtaining approximate network distance from the probing point in terms of Round Trip Time (RTT). Through statistical analysis we accurately estimate the number of intermediate hops to generate a rough map of the underlying network topology.

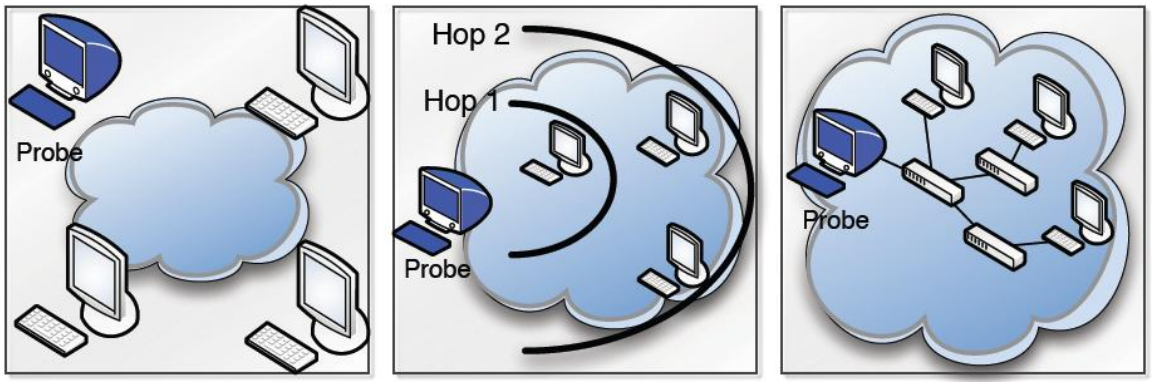


Figure 24: Exposing network structure from a vantage point.

The second phase (independent and complementary to the first phase) transmits specially crafted ARP packets to every host. These packets leverage updating of ARP forwarding table in ethernet switches to measure the common physical segments between pairs of probed hosts. At the end of the second phase, we have a relative distance map.

We combine the hop distances from the first phase with the map from the second phase to construct a complete map of the physical topology of the target network. We can do this because in an ethernet network point all networks can be modeled as trees [21]. Therefore, all paths from the probing point (root) to two different nodes (leaves) of the tree are Y-shaped. Knowing the distance from the root to each leaf (from the first phase) and the length of the shared part of the two connecting paths (from the second phase) we infer the complete topology.

5.1 Network Distance Inference

The first phase of our approach estimates network distance in terms of hops between the probing point and each reachable host. We measure the RTT of TCP ping packets and tune a linear model for pair-wise network distances. To initially enumerate reachable active hosts, we send a broadcast ARP packet to get their MAC addresses, and then send a standard ARP packet to each host to discover IP addresses. Our experiments demonstrate that larger packet sizes (i.e., Jumboframes [22] versus standard Ethernet frames) provide more accurate measurements.

Our experiments show that the sorted answering times of a single host have a typical pattern. This pattern has a central plateau representing steady state network behavior, with two outer zones for exceptionally fast/slow answering times (considered noise). We introduce a first-order derivative filter extracts the central plateau. This is shown in Figure 25. In the figure, there are four hosts with growing topological distance (hops). The blue area shows the data extracted by the filter. This reduces the average confidence interval (95%) from 12.30µs to 1.68µs (86.34% improvement).

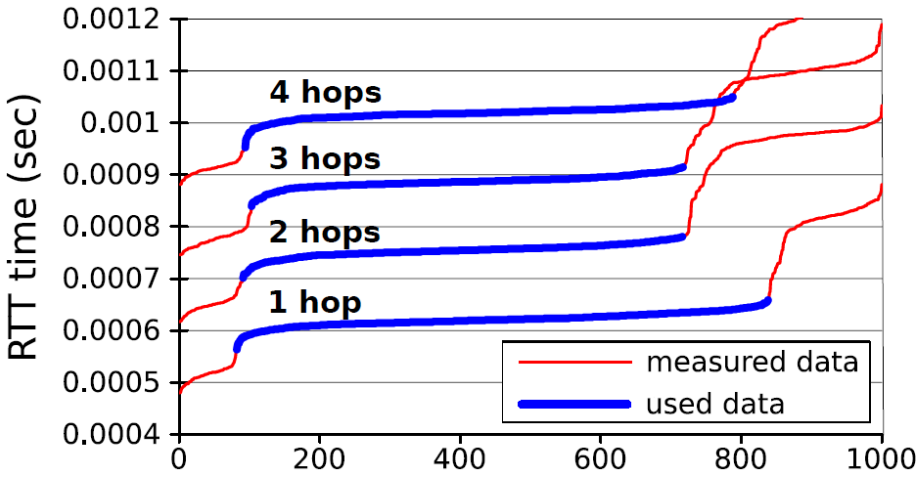


Figure 25: Plateau isolation filter improves estimation of network distances.

In general, the measured RTT between two hosts A and B is the sum of the time the packet spends in these places:

- Network interface A
- Peripheral switch connected to A
- Internal networking (unknown topology)
- Peripheral switch connected to B
- Network interface B

The probe has two network interfaces with known time for ping packet generation. We can leverage the probe timing information to compute switching time of any intermediate Layer-2 switching device to which it is connected. For directly connected switches, we use two local hosts (source and destination), with switching times through a simple difference.

We then estimate the intercept and slope of the line modeling the growth of the RTT, as shown in Figure 26. From this we infer actual distance: a measured RTT is compared to predicted RTTs and inherits the distance from the closest one. For all our test cases (in our laboratory and in Deterlab [23]), we correctly infer all distances where Jumboframe support is active.

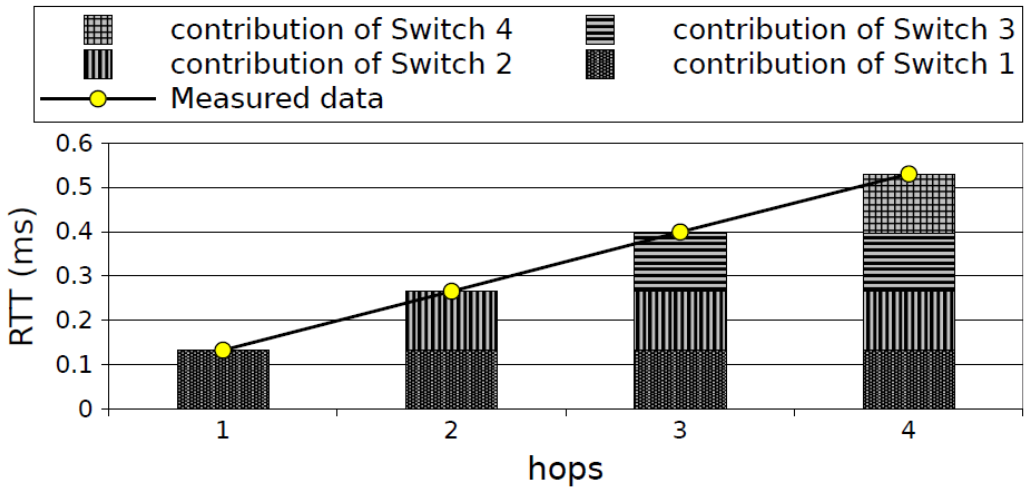


Figure 26: Inferring network distance from RTT growth model.

5.2 Shared Path Inference

The first phase of our topology discovery gives the distance from the probing point to each reachable host. But this alone is not enough to reconstruct the complete topology. We know a certain number of hops are needed to reach a host, but not their relative positions. Our second phase amends this and estimates approximate distances between host pairs. For two hosts A and B with hop distances D_A and D_B from the probe, we find parameter S_{AB} representing the degree of overlap between paths from probe to each host.

$S_{AB} = D_A = D_B$	Hosts A and B are connected to the same switch, and the two paths connecting them to the probe are completely overlapped.
$S_{AB} = min(D_A, D_B)$	One of the paths is a proper subset of the other one: to reach the farther host a packet sent by the Probe needs to reach the switch connecting the closer host and then continue its travel.
$S_{AB} < min(D_A, D_B)$	The two paths connecting the Probe to host A and B share S_{AB} hops and then split.
$S_{AB} = 1$	Only the first switch is shared by the two paths.

Figure 27: Parameter representing path overlap from probe to two hosts.

Figure 27 shows possible values of S_{AB} (non-zero because the switch to which the probe is plugged is shared by all paths). To measure S_{AB}, we inject two ARP streams: one is broadcast for ARP spoofing; the other is directed to a single host to sample network status.

As an example, consider the switch-based topology in Figure 28. We measure parameter S for host pairs Target to Host 1 (S_{TH1}) and Target to Host 2 (S_{TH2}).

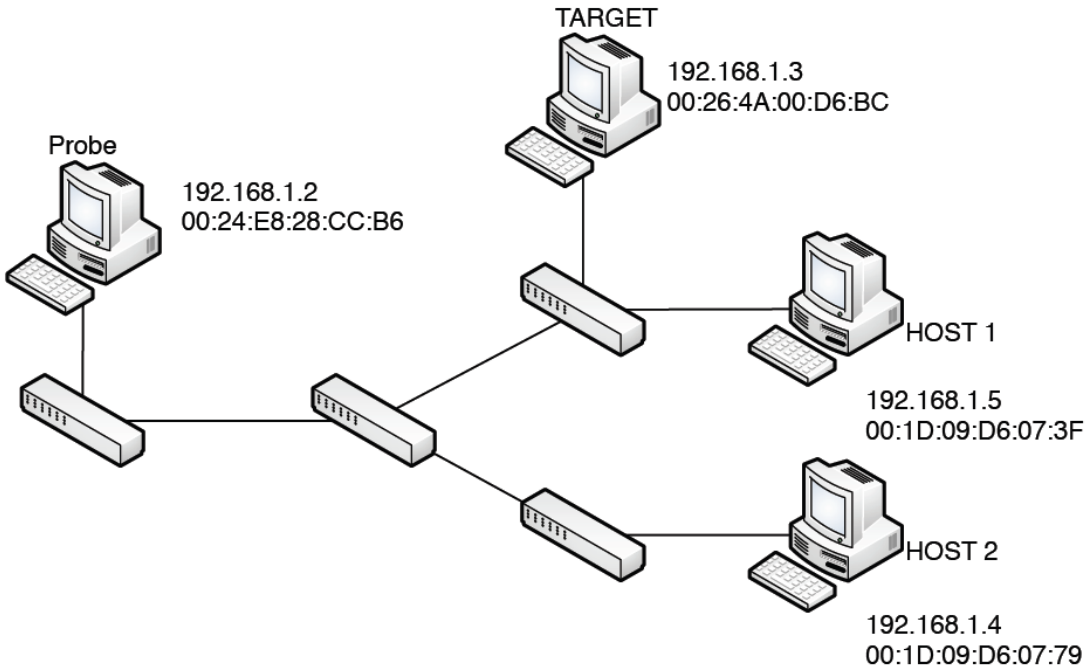


Figure 28: Example network for Layer-2 topology discovery.

As shown in Figure 29, the probe sends a broadcast request containing (as sender MAC) the address of the target host. As sender the probe uses its own real IP address, and as receiver it uses the IP address of the Target host. Since the network has only Layer-2 or Layer-3 switches, we use only the Ethernet header for forwarding and updating forwarding tables.

When this stream propagates through the network, all the crossed switches update their forwarding tables to forward the traffic directed to the Target host to the Probe (coherently with the broadcast Ethernet header). This affects only the switches along the path between the two hosts. When the target host sees the stream, it creates a stream of answers directed to a host that has its same MAC address but a different IP. These answers travel back to the probe, forcing the switches to again update their forwarding tables, restoring the initial condition.

Probe ARP broadcast request			
Ethernet	from	MAC	00:26:4A:00:D6:BC
	to	MAC	*broadcast*
ARP	from	MAC	00:26:4A:00:D6:BC
		IP	192.168.1.2
	to	MAC	*broadcast*
		IP	192.168.1.3
answer from Target host			
Ethernet	from	MAC	00:26:4A:00:D6:BC
	to	MAC	00:26:4A:00:D6:BC
ARP	from	MAC	00:26:4A:00:D6:BC
		IP	192.168.1.3
	to	MAC	00:26:4A:00:D6:BC
		IP	192.168.1.2

Figure 29: Broadcast ARP request and answer.

Figure 30 shows the final effects of this broadcast ARP stream on the forwarding tables of the switches in the target network. For a certain period of time, each switch along the path will forward packets directed to the target host to the probe. This Temporary Diverting Window (TDW) is minimal for the switch directly connected to the Target host, and grows linearly getting closer to the probe.

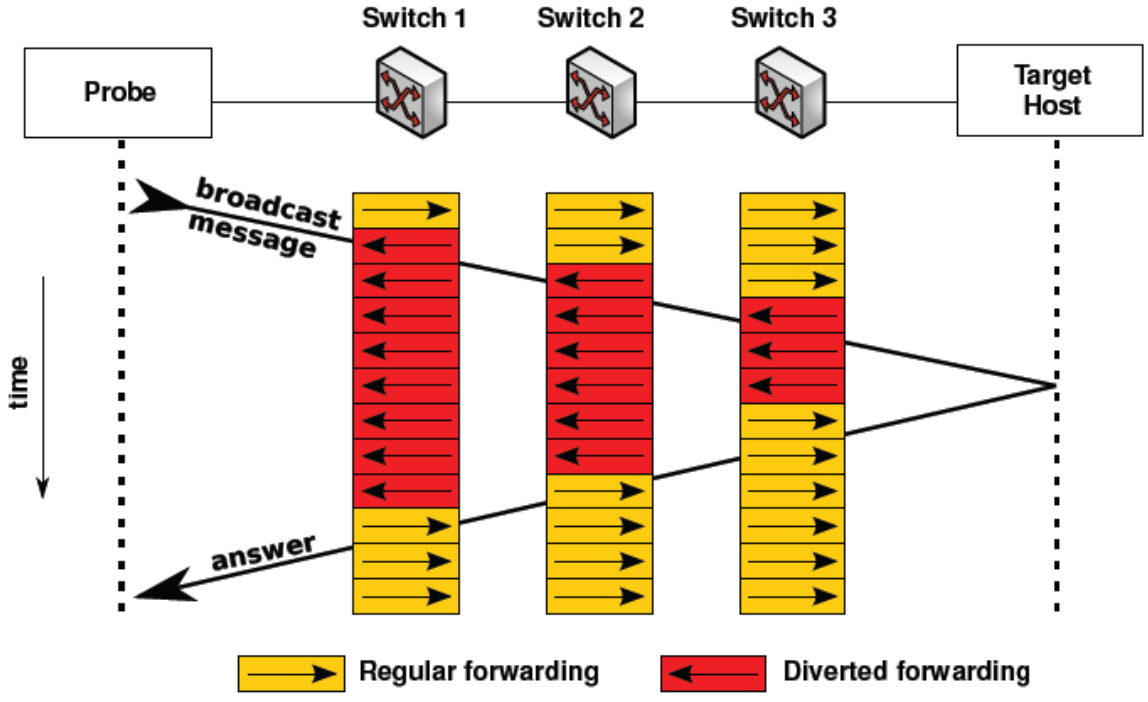


Figure 30: Effects of broadcast ARP stream on switch forwarding tables.

The second ARP stream measures the size of the TDW created by the first stream, to infer the shared part of the two paths. It is a series of ARP requests to another host, with target MAC as sender address for the ARP protocol and real probe MAC address in the ethernet header. Figure 31 shows the configuration of the stream to Host 1. Each packet generates an answer packet coming from Host 1. Since the answer packet must be delivered to a MAC address shared by both probe and target host, its actual forwarding path will depend on the state of the forwarding tables of the intermediate switches.

probe ARP direct request			
Ethernet	from	MAC	00:24:E8:28:CC:B6
	to	MAC	00:1D:09:D6:07:3F
ARP	from	MAC	00:26:4A:00:D6:BC
		IP	192.168.1.2
	to	MAC	00:1D:09:D6:07:3F
		IP	192.168.1.5
answer from Host 1			
Ethernet	from	MAC	00:1D:09:D6:07:3F
	to	MAC	00:26:4A:00:D6:BC
ARP	from	MAC	00:1D:09:D6:07:3F
		IP	192.168.1.5
	to	MAC	00:26:4A:00:D6:BC
		IP	192.168.1.2

Figure 31: ARP direct request and answer.

If the second stream is directed to a direct neighbor of the target (e.g., Host 1), the vast majority of answers will be forwarded to the target, since the TDW on the last switch is very small. If it is directed to a host closer to the probe (e.g., Host 2) a larger number of packets will be diverted to the probe. This stream of packets is a form of sampling, i.e., each packet gives information about the status of the temporal window.

6. NETWORK MONITORING AND CHANGE DETECTION

Continuous improvements in the forwarding capacity of Layer-2 switching devices are re-shaping the networking landscape and present new challenges to black-box network inference. Multitudes of endpoint devices such as laptops to smart-phones are connected to target networks using switching infrastructure. This makes network topology discovery and monitoring even more challenging, but provides new opportunities for network attack.

We describe scalable fingerprinting mechanisms that can monitor network topology changes, detect new active devices, and identify changes in available capacity of target networks [24]. These mechanisms can also automatically generate the connectivity mapping of large-scale network topologies and report any modifications. Efficiency and

accuracy of these mechanisms have been experimentally validated on mixed-speed and mixed-vendor networks.

Our goal is to measure various aspects of a target network state automatically and periodically identify changes in the underlying network topology, network devices, end-point systems, and network capacity. In our approach, detecting changes is a matter of simple statistical analysis, showing deviation from a baseline state.

We extract topological information from the network through measuring, for any two hosts, how the paths connecting them to a network probe relate to each other. Two streams of ARP packets are crafted to sample the network state. Our technique of triangular sampling allows us to create a snapshot of the current topological state of the network. The extracted information can be used to monitor evolution of a target network.

6.1 Triangular Sampling

We perform a triangular sampling operation to gather information about the network topology. Each triangular sampling produces two streams of ARP packets, and explores the network connectivity between the probe and two other hosts. We perform the samplings on any two active hosts: one host acts as Target (T) and the other one as Support (S).

Each sampling is independent from the others, allowing us to break the measuring process into several sessions for maintaining stealth. To perform a single triangular sampling two ARP streams are injected in the network: one is broadcast and performs an ARP manipulation technique focused on target host T, the other one is a stream of gratuitous ARPs directed to S and is used to sample the status of the network.

The first ARP stream is a broadcast request containing, as sender MAC, the address of the target host. When the stream propagates through the network all the crossed switches update their forwarding tables to redirect the traffic addressed to the target host to the probe, coherently with the broadcast Ethernet header. That change actually affects only the switches along the path between the two hosts, since all other hosts need to pass through the connecting path anyway.

The stream reaches the target host it answers with a stream directed to a host that has its same MAC address, but a different IP. This stream of answers travels back to the probe, and forces all the switches to update again their forwarding tables, restoring the initial condition. Each switch along the path, for a certain period of time, will forward the packets directed to the target host to the probe. This diversion window is minimal for the switch directly connected to the target host, and grows getting closer to the probe.

The second stream is used to sample the size of the diversion window created by the first stream. It consists of a series of gratuitous ARP requests to another host, using the target host MAC as sender address for the ARP protocol, and using the real probe MAC address in the Ethernet header. Since the answer packet must be delivered to a MAC address which is shared by the probe and the target host, its actual forwarding path will depend on the state of the forwarding tables of the intermediate switches.

If the second stream is addressed to a direct neighbor of the target host, the vast majority of the answers will be forwarded to the target host, since the diversion window on the last switch is minimal. If the second stream targets a host closer to the probe, a bigger number of packets will be diverted to the probe. Our triangular samplings wield information on the current topology status. In particular, it can be used to infer the extent of overlap in the paths connecting the probe to any other two hosts.

Triangular samplings modify network connectivity for a brief period of time. This change is very limited, especially with guaranteed delivery protocols such as TCP. A single triangular sampling lasts for a RTT between the probe and a target host. Of that time, only a fraction causes actual connectivity disruption.

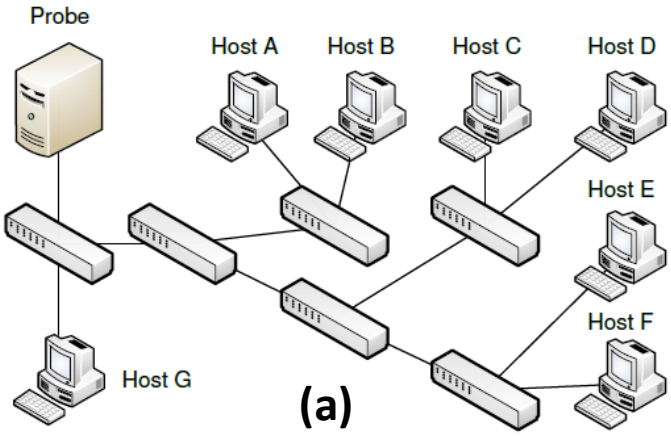
Still, we introduce an extra safeguard for ensuring stealth. If the probe starts receiving regular traffic addressed to the target host (or if it receives more than a fixed amount of packets), it immediately stops testing. In our experiments, with this safeguard in place there was no measurable disruption in connectivity.

On average, we observe generated network traffic of 16 Kbps during triangular samplings. From the network point of view this is negligible overhead on the target network infrastructure.

6.2 Topology Reconstruction

We introduce the network distance matrix, which is an alternate representation of a network tree topology. We can infer the distance matrix from triangular samplings for network topology reconstruction.

In a given tree topology, with a chosen root node, we can define the distance from the root to any leaf node. Intuitively, this distance represents the number of network devices that need to be crossed by a packet traveling from root to leaf. The two paths connecting the root to two nodes will overlap to some degree. We measure the common part of the two paths, for any two hosts, in a symmetric distance matrix. Figure 32 shows a distance matrix for an example network.



	A	B	C	D	E	F	G
A	-	3	2	2	2	2	1
B	3	-	2	2	2	2	1
C	2	2	-	4	3	3	1
D	2	2	4	-	3	3	1
E	2	2	3	3	-	4	1
F	2	2	3	3	4	-	1
G	1	1	1	1	1	1	-

(b)

Figure 32: Example network topology (a) and its distance matrix (b).

The distance matrix is unique for any tree topology, and we can build one starting from the other. In particular, our probes measure host distances, yielding the distance matrix, from which we infer the network topology.

Triangular samplings have a direct relationship with the distance matrix. Figure 33 shows the triangular samplings from a sample network topology. The samplings tend to cluster, and in a homogeneous network follow a linear model. Our experimental evidence shows that hierarchical clustering performs better in separating the values for inferring the distance matrix from the matrix of all triangular samplings.

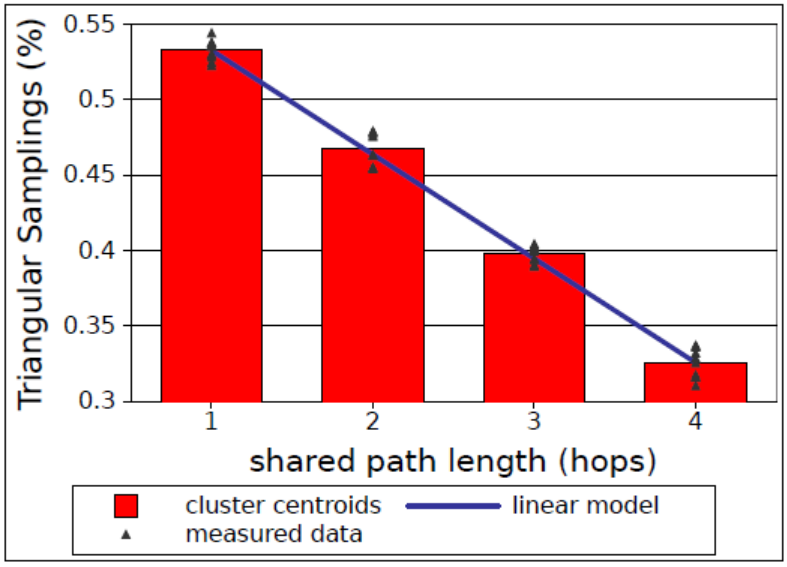


Figure 33: Clustering triangular samplings.

A complete sampling of the whole network requires a triangular sample for each possible pair of hosts. The time required for a full network exploration grows quadratically with the number of hosts. A single triangular sampling takes 15-20 seconds, depending on network speed. With 100 hosts, a naïve complete sampling would take about 27 hours.

To reduce the number of tests, we introduce a technique that eliminates the intrinsic redundancy of full network explorations and allows us to extract topology information without doing a full host-to-host sampling.

Hosts connected to the same switch share the same topological information. Each of these "island" hosts can be considered a single entity made by homogeneous components. All hosts from a specific island will expose a similar behavior toward triangular samplings, since only the topological position influences the tests. We can then select a single host from each island and use it as a representative for all other hosts sharing the same switch.

The actual gain of this method depends on host sampling order and on the specific topology shape. Flat topologies with many hosts per island will allow a great reduction

in the number of required samples. Very vertical topologies with many small islands will limit its effectiveness. For an experimental network with 30 switches and 90 hosts we measure an effective reduction of triangular samplings between 534% and 796%.

6.3 Experiments

The target of our experiments is to validate the ability to correctly infer the distance matrix, since it is equivalent to the network topology.

In our experiments, we use the Mininet [25] on a Dell PowerEdge R715 server with 128 GB of Ram and two 12 core CPUs. The Mininet environment facilitates the implementation and testing of new ideas in network research. It exploits Linux kernel name-spaces to simulate hosts with their own network stack. Mininet exposes a python software interface to define the networks, which allow us to easily implement large scale topologies for testing.

In our experiments we simulate networks with up to 30 switches and up to 90 hosts. Figure 34 shows a sample topology from our experiments. Our test topologies have three hosts per switch. We classify the topologies on their depth, i.e., the maximum distance of a host from the probe.

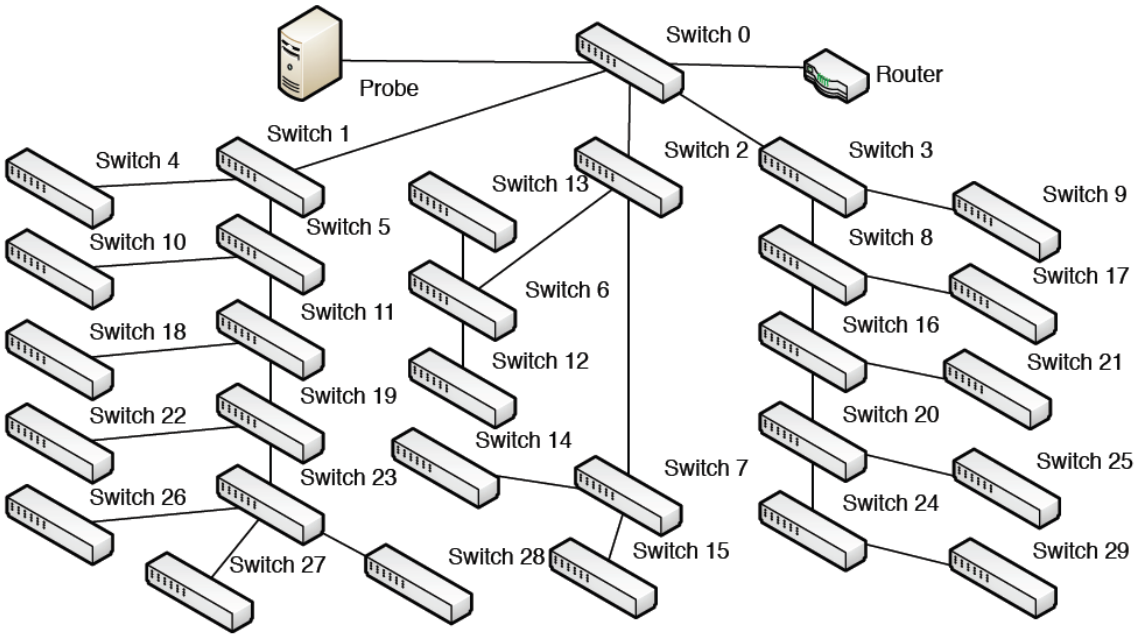


Figure 34: Example network topology for testing.

Figure 35 shows the averages of the clustering correctness for several values of network depth and number of hosts. The lines marked as linear indicate a simple line topology, where no branches were present. This topology is the simplest configuration, which we use as an experimental reference. For nonlinear topologies, we test different configurations and report the averages of clustering correctness.

Network depth	Hosts number	Correctness (%)	Linear
3	21	100.00	✓
4	27	100.00	
5	33	100.00	
6	18	100.00	✓
6	36	98.76	
6	60	96.76	
7	21	100.00	✓
7	54	95.85	
7	90	89.76	

Figure 35: Clustering correctness for various network sizes.

Our approach correctly detects all the tested network topologies up to depth five. At depth six there are a few clustering errors, caused by a less sharp division in triangular samplings values. Still, with very wide topologies of depth seven we limit errors to about 10% of the network hosts.

7. MULTI-STEP ATTACK GRAPH MODELING

As described in Section 3.4, we passively infer communications to sets of hosts in a local segment, and use that information to build a model of the target network for attack graph analysis.

In particular, as an initial model, we use the TTL field of IP packets to distinguish hosts as either inside or outside the local network segment of a passively observing agent. In this case, the agent observes 3,833 packets over a period of 11 minutes, from which we infer 119 hosts on the local segment and 47 outside hosts. Communication between a pair of inside and outside hosts is modeled as a connection to a potentially vulnerable client or service on the inside host.

The resulting attack graph in Figure 36 shows the attack paths from outside hosts into hosts in the target network. The summary view in Figure 36(a) shows 24 different attack vectors into the target network. Expanding to the view in Figure 36(b), this shows that all 24 potential attack vectors are against a single target host.

There are 118 additional hosts in the target network (aggregated to a single graph node), but none of those are exposed directly from the outside. This initial model does not distinguish whether outside hosts are actually in the same or different segments, i.e., all outside hosts are placed in a common protection domain.

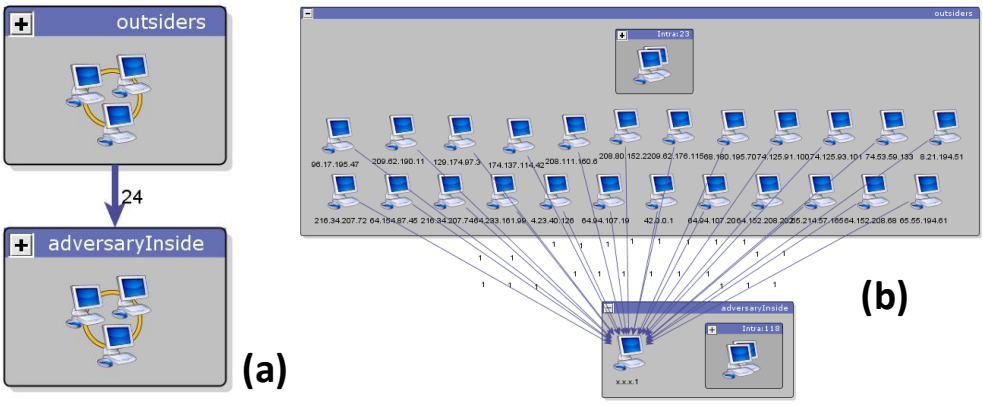


Figure 36: Attack graph showing outsiders attacking insiders.

For a more refined attack graph model, we use TTL values to infer traffic as originating in separate (versus a common) source segments. The resulting attack graph shows potential attackers as starting from different network segments (modeled as protection domains), based on differing TTL values from the source hosts.

This relies on the fact that operating systems generally set a packet's initial TTL to a known value, e.g., 64, 128, or 255. Network devices decrement TTL, so we use TTL to infer the number of hops from the initial source host. In this way we group hosts by common TTL values (more specifically, common deltas from default values). This is necessary but not sufficient for accurately predicting hosts in the same segment, though we could further group by IP address similarities. Figure 37 is the histogram of observed values TTL, e.g., showing large number of packets with TTL = 128 (originating from the local segment).

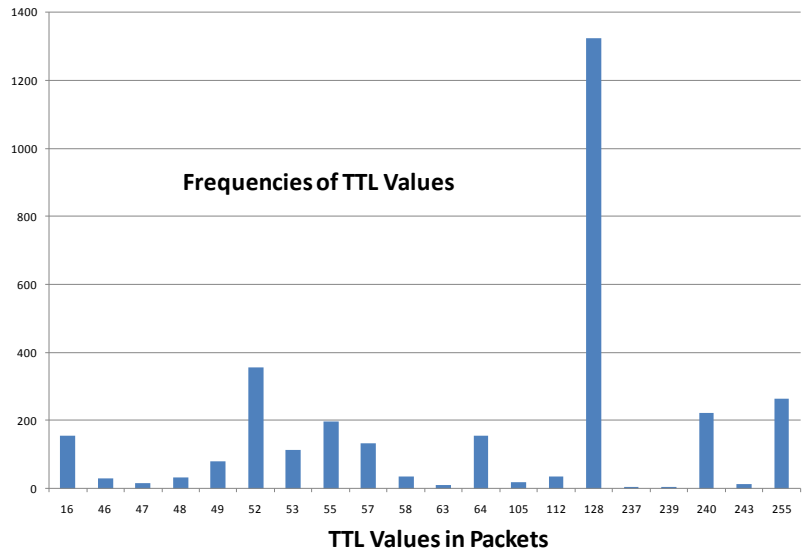


Figure 37: Histogram of observed TTL values.

In this more refined analysis of TTL values, if a host sends a packet with known (powers of 2) initial value, we infer that it is in the local segment of the inside observing agent. In this case, we infer 119 inside hosts.

Outside hosts (47 inferred hosts) that send packets to inside hosts are then grouped in separate protection domains (14 inferred outside domains) according to common source TTL values. These outside hosts are modeled as having connections to potentially vulnerable software on an inside host, i.e., in our database of modeled attacker exploitations. Figure 38 shows the resulting network model, which we use as input to multi-step attack graph analysis.

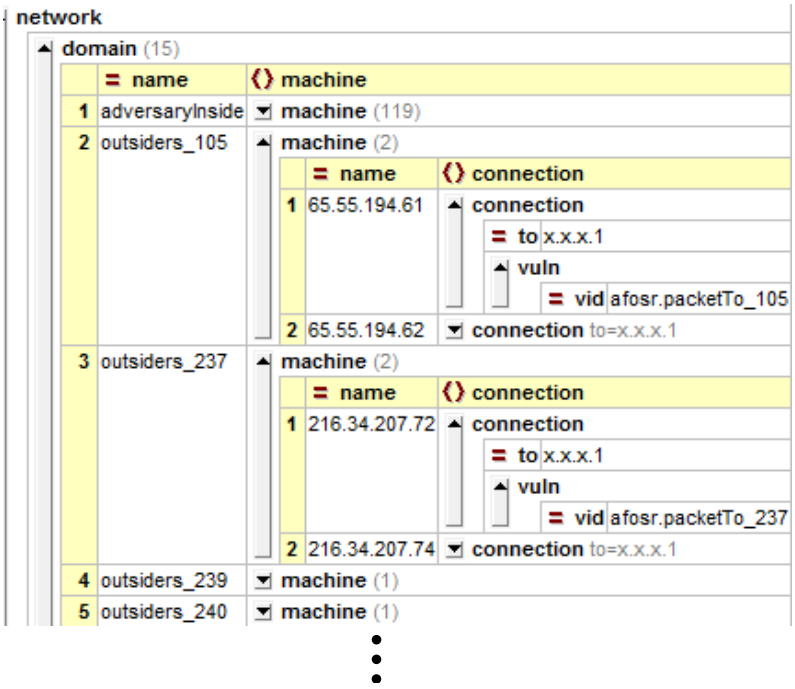


Figure 38: Network model with attackers in multiple segments.

Figure 39 shows the attack graph resulting from multi-step attack analysis of the network model in Figure 38. This shows potential attacks originating from each of the outside protection domains, where these domains are based on our TTL analysis. From the top of the figure, this shows that there are 7 outside segments that each has a single attack possible against the target segment. Similarly, from the bottom of the figure, there are 4 outside segments that each has 2 possible attacks against the target, and 3 outside segments that each has 3 possible attacks against the target.

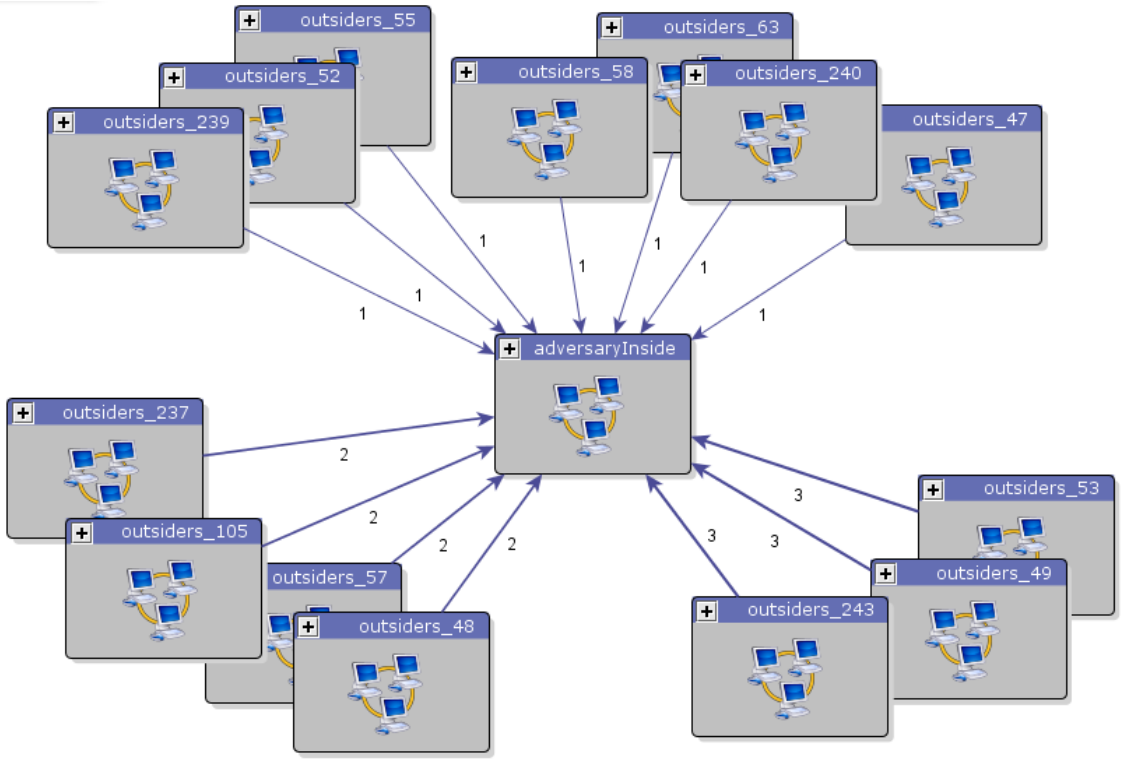


Figure 39: Attack graph with attackers in multiple segments.

In this attack graph, still only single-step attacks appear, although from multiple origins. This is a result of deploying only a single inside observing agent, which has a limited view of network traffic. That is, TTL values only give the number of network hops between source hosts and the observing agent.

Vantage points in multiple segments are needed for inferring potential attacks among other segments. Thus, multiple inside agents deployed in different segments of a target network yield more complex multi-steep attack graphs.

Figure 40 shows our full-featured tool for interactive attack graph analysis and visualization. This tool was developed for active scanning of white-box networks, i.e., for network attack defense [26][27][28][29][30]. In this project, we populate our attack graph tool via passive and stealthy active inference of black-box networks, for strategic attack planning.

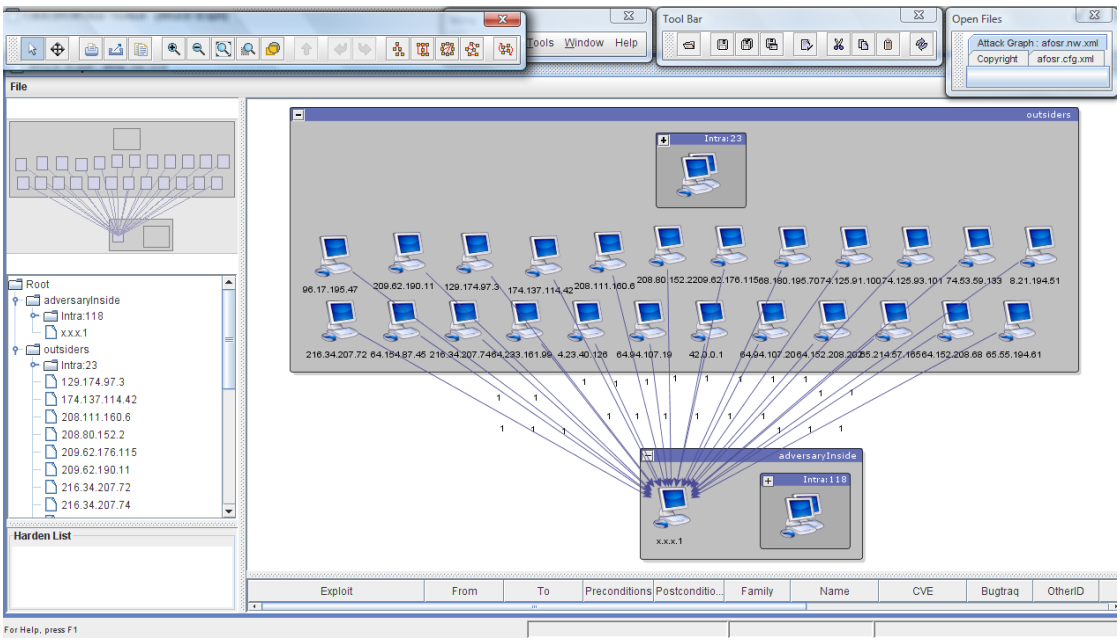


Figure 40: Tool for attack-graph based strategic planning.

8. REFERENCES

1. J. Kuntzelman, *Comparative Analysis of Active and Passive Mapping Techniques in an Internet-Based Local Area Network*, Master's Thesis, Air Force Institute of Technology, 2004.
2. S. Webster, R. Lippmann, M. Zissman, "Experience Using Active and Passive Mapping for Network Situational Awareness," in *Proceedings of the 5th IEEE International Symposium on Network Computing and Applications*, 2006.
3. Y. Thomas, H. Debar, B. Morin, "Improving Security Management through Passive Network Observation," in *Proceedings of the First International Conference on Availability*, Reliability and Security, 2006.
4. G. Bartlett, J. Heidemann, C. Papadopoulos, "Understanding Passive and Active Service Discovery," in Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement, 2007.
5. J. Treurniet, *An Overview of Passive Information Gathering Techniques for Network Security*, Technical Memorandum DRDC-OTTAWA-TM-2004-073, Defence R&D Canada – Ottawa, 2004.
6. R. Deraison, R. Gula, T. Hayton, "Passive Vulnerability Scanning," white paper, http://www.tenablesecurity.com/images/pdfs/passive_scanning_tenable.pdf, 2007.
7. M. Zalewski, p0f Passive OS Fingerprinting Tool, http://lcamtuf.coredump.cx/p0f.shtml, 2006.
8. Ettercap, http://ettercap.sourceforge.net/.

9. Snort Intrusion Detection System, http://www.snort.org/.
10. R. Lippmann, D. Fried, K. Piwowarski, W. Streilein, "Passive Operating System Identification from TCP/IP Packet Headers," in *Proceedings of the ICDM Workshop on Data Mining for Computer Security*, 2003.
11. *Description of the Microsoft Computer Browser Service*, Microsoft Knowledge Base article, http://support.microsoft.com/kb/188001.
12. *GetVersionEx: Windows Version Info (Wrapper Routines)*, http://vbnet.mvps.org/index.html?code/helpers/iswinversion.htm.
13. *List of User-Agents*, http://www.user-agents.org/.
14. Nessus Vulnerability Scanner, http://www.nessus.org/nessus/.
15. K. Kortchinsky, "Making Windows Exploits More Reliable," Blackhat Europe, 2007.
16. *Remote Desktop Protocol*, Wikipedia, http://en.wikipedia.org/wiki/Remote_Desktop_Protocol.
17. *OSI Model*, Wikipedia, http://en.wikipedia.org/wiki/OSI_model.
18. J. Almillategui, N. Nassicari, A. Stavrou, S. Jajodia, *Switchwall: Automated Ethernet Topology Discovery*, Technical Report GMU-CSIS-TR-2010-1, Center for Secure Information Systems at George Mason University, 2010.
19. *Simple Network Management Protocol (SNMP) Vulnerabilities Frequently Asked Questions (FAQ)*, Carnegie Mellon Software Engineering Institute, http://www.cert.org/tech_tips/snmp_faq.html.
20. *An Ethernet Address Resolution Protocol*, http://www.ietf.org/rfc/rfc826.txt.
21. *IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Bridges*, http://standards.ieee.org/getieee802/download/802.1D-2004.pdf.
22. P. Dykstra, *Gigabit Ethernet Jumbo Frames*, white paper, WareOnEarth Communications, 1999.
23. *Deterlab Network Security Testbed*, http://www.isi.deterlab.net/.
24. N. Nassicari, J. Almillategui, A. Stavrou, S. Jajodia, "Switchwall: Automated Network Fingerprinting and Behavior Deviation Identification," in *Proceedings of the 31st Annual IEEE International Conference on Computer Communications*, to appear.
25. *Mininet: Rapid Prototyping for Software Defined Networks*, http://yuba.stanford.edu/foswiki/bin/view/OpenFlow/Mininet.
26. S. Noel, "Network Assessment Tool Demonstration," Air Force Intelligence, Surveillance, and Reconnaissance Conference, January 2010.
27. S. Noel, S. Jajodia, "Advanced Vulnerability Analysis and Intrusion Detection through Predictive Attack Graphs," Workshop on Critical Issues in C4I, May 2009.

28. S. Noel, M. Elder, S. Jajodia, P. Kalapa, S. O'Hare, K. Prole, "Advances in Topological Vulnerability Analysis," Cybersecurity Applications & Technology Conference for Homeland Security, March 2009.

29. S. Jajodia, S. Noel, "Topological Vulnerability Analysis," Army Research Office Workshop on Cyber Situational Awareness, 2009.

30. S. Noel, S. Jajodia, "Proactive Intrusion Prevention and Response via Attack Graphs," in *Practical Intrusion Analysis: Prevention and Detection for the Twenty-First Century*, 2009.