Direct Network Access and Market Applications

Andrew Rioux

October, 20 2025

1 Direct Network Access

Direct network access is a technique that can be used to decrease the likelihood of detecting malicious network traffic on an endpoint. It can be seen as an extension of the MITRE ATT&CK technique T1205 Traffic Signaling, as it allows a threat actor to either respond to signaled traffic or to initiate traffic.

1.1 Technique

The basis of direct network access is a raw network socket. Linux provides the opportunity to create raw sockets with the standard `socket` system call, BSD based systems such as FreeBSD and OpenBSD allow opening BPF sockets, and Windows has `SOCK_RAW` sockets that can be used in the kernel. Linux and BSD based systems do not require kernel modification, allowing code to malicious operate entirely in userspace. With raw sockets, it is possible to re-implement Ethernet based protocols such as ARP and IPv4 and go farther to implement TCP or UDP. Re-implementing almost the entire OSI model has downsides such as malware size and complexity or distinct signatures. However, it can also provide enormous benefits for an adversary if defenders are unprepared or unable to recognize the indications of compromise.

1.2 Tools

The most difficult technical challenges to developing a market application implementing this technique are to handle creation and management of raw sockets and to re-implement TCP/IP. Several libraries exist for both cases. The standard library to handle socket management would be libraries making use of the `libpcap` API, such as `libpcap`, `winpcap`, or `npcap`. Likewise, there are several libraries to re-implement TCP/IP such as lwIP (C), smoltcp (Rust), or tcpip (Go).

1.3 Limitations and Detections

Implementing a second network stack on a currently running machine can provide complications. UDP is much simpler, and direct network access can be used to send and receive UDP traffic with the host kernel at most replying with ICMP traffic to indicate that the port is closed. TCP presents some more technical challenges, since a host kernel may receive TCP data that it is not expecting that is intended for the alternate network stack. In response, the host kernel will follow the TCP RFC and respond with a RST packet. To circumvent this, a stateful firewall on the host can be used that only allows response traffic to outbound connections initiated by the kernel, the program can use IP address spoofing, or the program can use IP and MAC address spoofing.

Direct network access can be difficult to detect at a network level, as the packets can be well-formed and standards compliant. Direct network access will need some level of host based detection and correlation with network traffic.

On Linux, `/proc/net/raw` will list all the currently open raw sockets, and auditd can be used to listen for the use of the `socket` system call with the `SOCK_RAW` parameter.

On FreeBSD, software actively using `libpcap` to perform direct network access can be enumerated by querying for BPF filters in place.

Windows does not provide userspace tools to create network sockets, requiring a driver component. Identifying unauthorized drivers that have been installed and loaded, particularly network based drivers can be a useful indicator of direct network access attempts.

1.4 Applications

The kernel of an operating system provides a network firewall and auditing of network connections. Direct Network Access operates on a different network stack, enabling an adversary to ignore the local firewall and create network connections that are not logged by the local system. IP addresses and MAC addresses are configuration options to set, allowing for easy MAC or IP spoofing.

1.5 Differences from Traffic Signaling

Traffic signaling focuses on using `libpcap` or raw sockets to receive data and move on to later stages of operation. Direct network access allows hiding all network traffic from the host kernel, and more importantly allows initiation of traffic for C2 callbacks.

2 Market Application: Sparse

Sparse as an application comes in two versions, with the first being a bind shell built on UDP and the second being a C2 beacon using HTTP and TLS. Both make use of direct network access to negate the local firewall and standard network socket auditing. Sparse version 1 acts in a similar manner to SSH, allowing an adversary to specify a server to connect to and a key file. The client then establishes a secure connection to the server, with the bind shell able to run as a Windows Service, a Windows non-service executable, a Linux ELF executable, or a FreeBSD ELF executable.

Sparse version 2 makes use of TCP and initiates C2 callbacks with IP and MAC spoofing to hide the computer performing the callback. Sparse version 2 makes use of other process hiding capabilities such as modifying DLLs or shared objects to embed itself in already existing processes, enabling an adversary to perform remote actions on an endpoint without a separate process existing or any local network logs appearing. With version 2 being a C2 beacon, it also disables its raw socket when not performing callbacks to not appear during audits.

Source code for Sparse version 1 can be found here: `https://gitea.riouxs.co/andrew.rioux/sparse`, and source code for version 2 can be found here: `https://gitea.riouxs.co/andrew.rioux/sparse-v2`. Both source code repositories include documentation on how to build the software and deploy it.

3 Potential Future Applications: NAT

With a threat actor being able to arbitrarily read and write data to the network, installing a piece of malware that makes use of this technique on a network firewall can have significant consequences. If successfully implemented on the firewall, properly engineered malware can be programed to bridge between two networks in a way that bypasses network level restrictions and auditing. However, a more serious concern could arise from a threat actor engineering malware to perform Network Address Translation or port forwarding by listening to an internal zone and a WAN zone. This could enable threat actors to have persistent access to internal services, or allow threat actors to use a different IP subnet on the internal network and hide the traffic of their C2 beacons from the regular firewall kernel.