# ShadowMove: A Stealthy Lateral Movement Strategy

Amirreza Niakanlahiji*
*University of Illinois Springfield*
*aniak2@uis.edu*

Jinpeng Wei
*UNC Charlotte*
*jwei8@uncc.edu*

Md Rabbi Alam
*UNC Charlotte*
*malam5@uncc.edu*

Qingyang Wang
*Louisiana State University*
*qwang26@lsu.edu*

Bei-Tseng Chu
*UNC Charlotte*
*billchu@uncc.edu*

## Abstract

Advanced Persistence Threat (APT) attacks use various strategies and techniques to move laterally within an enterprise environment; however, the existing strategies and techniques have limitations such as requiring elevated permissions, creating new connections, performing new authentications, or requiring process injections. Based on these characteristics, many host and network-based solutions have been proposed to prevent or detect such lateral movement attempts. In this paper, we present a novel stealthy lateral movement strategy, `ShadowMove`, in which only established connections between systems in an enterprise network are misused for lateral movements. It has a set of unique features such as requiring no elevated privilege, no new connection, no extra authentication, and no process injection, which makes it stealthy against state-of-the-art detection mechanisms. ShadowMove is enabled by a novel *socket duplication* approach that allows a malicious process to silently abuse TCP connections established by benign processes. We design and implement `ShadowMove` for current Windows and Linux operating systems. To validate the feasibility of ShadowMove, we build several prototypes that successfully hijack three kinds of enterprise protocols, FTP, Microsoft SQL, and Window Remote Management, to perform lateral movement actions such as copying malware to the next target machine and launching malware on the target machine. We also confirm that our prototypes cannot be detected by existing host and network-based solutions, such as five top-notch anti-virus products (McAfee, Norton, Webroot, Bitdefender, and Windows Defender), four IDSes (Snort, OSSEC, Osquery, and Wazuh), and two Endpoint Detection and Response systems (CrowdStrike Falcon Prevent and Cisco AMP).

## 1 Introduction

Advanced Persistent Threats (APTs) are sophisticated, well-planned, and multistep cyber attacks against high profile targets such as government agencies or large enterprises. Such attacks are conducted by groups of well-resourced knowledgeable attackers (such as Lazarus or APT38) and cost companies and government agencies billions of dollars in financial losses per year [28].

APT attackers commonly use spearphishing or watering hole attacks to find a foothold within target networks. Once they entered the target networks, they cautiously use the compromised systems as stepping stones to reach other systems until they get access to the critical systems, such as file server containing confidential documents, buried deep inside the networks; this incremental movement toward the critical systems is called *lateral movement*.

Lateral movement can be achieved in a number of ways. Attackers can exploit vulnerabilities in network services, such as SMB or RDP, to laterally move across networks. However, due to advances in defense mechanisms, finding such vulnerabilities and successfully exploiting them without being detected has become increasingly difficult. Alternatively, attackers can harvest user credentials from compromised systems and reuse such credentials to perform lateral movement (e.g., credential dumping [43], pass-the-hash, or pass-the-ticket [24–26, 37, 38]). However, this approach requires new network connections to be created and thus can be detected by network-level defenses if the new connection deviates from the normal communication pattern among legitimate systems [34, 35, 51]. Using another approach, adversaries can employ hijacking attacks that modify a legitimate client in order to reuse its connection for lateral movement (e.g., by patching a SSH client to communicate with the SSH server without knowing the password [19]). However, such attacks are application- and protocol- specific and require process injection; they are hard to implement and prone to detection as existing host-based defensive solutions (e.g., Windows Defender ATP [48]) recognize various process injection techniques.

In this paper, we present a novel lateral movement strategy, called ShadowMove, which enables APT attackers to move stealthily among the systems in enterprise networks without being discovered by existing host-level and network-level de-

*Part of this research was performed while being a Ph.D. student at UNC Charlotte fensive mechanisms as demonstrated in Section 5. We assume that attackers want to avoid exploiting vulnerabilities in remote services during their operation to reduce the chance of being exposed by intrusion detection systems (IDSes). In this attack scenario, attackers passively observe communication dynamics of the compromised systems to gradually construct their model of normal behaviors in the target network and utilize this model to choose the next victim system. Moreover, to make the attack even stealthier, attackers restrict themselves to only *reuse established connections*. Many application protocols such as WinRM (Windows Remote Management) and FTP allow users to perform some operations on the remote server. Attackers inject their own commands in the command streams of such protocols to achieve their goal. For example, attackers can execute a program remotely by injecting commands in an established WinRM session (Section 4.4), or they can inspect the file system on the remote system by injecting FTP commands in an established FTP connection (Section 4.2).

ShadowMove does not use any code in benign client processes to inject fabricated commands. Instead, it employs a novel technique to secretly duplicate sockets owned by legitimate clients and injects commands through such stolen sockets (Section 3.4). By doing so, no new connection will be created and no new authentication will be performed as the injected commands are interpreted in the context of already established sessions; this means that the attacker does not need to pass any authentication.

In this work, we show how an attacker can implement such an attack on a typical enterprise network. To this end, we develop a prototype system that can hijack existing TCP connections established by an FTP client (Section 4.2), a Microsoft SQL client (Section 4.3), and a WinRM client (Section 4.4) running under the same user account as our prototype and without any elevated privileges. We also present a Prolog-based planner that an attacker can utilize to systematically plan for lateral movement by hijacking available connections. In this way, the attacker can reach the critical systems significantly stealthier than existing attack scenarios. We discuss the technical challenges on how attackers can inject their packets that conform to the protocol running over an established TCP connection and be acceptable to the server on the other end of the connection.

We summarize our contributions as follows:

- We present a new class of lateral movements which is completely undetectable by existing network and host-based defensive solutions including IDSes, Antivirus, and EDR (Endpoint Detection and Response) systems.
- We propose a novel socket duplication technique that enables attackers to reuse connections established by other processes on a compromised system. We, then, develop a lateral movement framework on top of this technique.
- We demonstrate the feasibility of our idea by building a prototype system on Windows 10 that successfully hijacks FTP, TDS (used by Microsoft SQL Server), and WinRM connections for lateral movements. This Windows prototype demonstrates all features of ShadowMove, requiring no elevated privilege, no new connection, no extra authentication, and no process injection. We also build a prototype that successfully hijacks FTP on Ubuntu 18.04 without requiring elevated privilege, new connections, or extra authentication. However, the design is not as stealthy as its Windows counterpart because it relies on process injection and requires stronger assumptions about the attacker (Section 3.4.3).
- We experimentally confirm that our prototypes can evade the detection of five top-notch anti-virus products (McAfee, Norton, Webroot, Bitdefender, and Windows Defender), four IDSes (Snort, OSSEC, Osquery, and Wazuh), and two emerging Endpoint Detection and Response systems: CrowdStrike Falcon Prevent and Cisco AMP. It is important to point out that CrowdStrike Falcon Prevent is known to detect lateral movements.

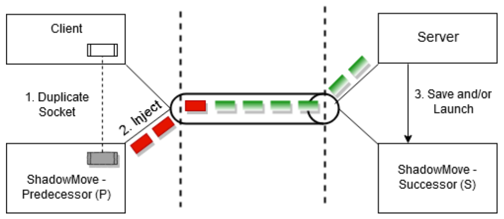


Figure 1: ShadowMove Lateral Movement

The result of our study calls for a revisit of enterprise protocols in terms of their susceptibility to hijacking attacks.

## 2 ShadowMove Approach

The basic idea of ShadowMove is to reuse established and legitimate connections to laterally move within the compromised network. As shown in Figure 1, ShadowMove works in three main steps: first, it silently duplicates a socket used by a legitimate client application to communicate with a server application; second, it uses the duplicated socket to inject packets in the existing TCP session between the client and the server; third, the server handles the injected packets and unintentionally saves and/or launches a new instance of ShadowMove. As a result of these steps, an attacker stealthily moves from the client machine to the server machine.

Since ShadowMove restricts itself to reuse established connections to neighboring systems, it can ensure intrusion detection systems that raise alarms for unexpected connections cannot detect its operation. Moreover, by doing so, the attack can bypass the authentication phase required for establishing a new connection. ShadowMove attack is noteworthy from both a host security perspective and a network security perspective: at the host level, ShadowMove abuses resources owned by a victim process (i.e., established and authenticated network connections); on the other hand, because what ShadowMove abuses are sockets, its attack actions extend to the network level, by blending malicious network traffic with benign network traffic.

## 2.1 Fundamental Weaknesses Exploited by ShadowMove

Two fundamental weaknesses in the existing computing environment enable ShadowMove attacks. The first weakness stems from the two conflicting but essential requirements, namely process isolation and resource sharing, in commodity operating systems such as GNU Linux and Microsoft Windows. The next weakness arises from the fact that many of the existing networking protocols lack proper built-in message origin integrity validation mechanisms, which makes them susceptible to message injection attacks.

Process isolation and process (resource) sharing are conflicting requirements. A process has a virtual address space, open handles to system objects, and other attributes. All processes in an operating system must be protected from each other's activities, for reliability and security reasons [52]. The protection mechanism of a modern OS isolates the access to different kinds of resources (e.g., CPU, memory, and I/O devices) among processes. For example, memory isolation puts each process into its own "address space". On the other hand, modern OSes support sharing among processes because sharing of data/resources can be useful. Take *socket sharing* for example, one process first creates sockets and establishes connections, then it hands off those sockets to other processes that will be in charge of information exchange through those sockets. However, sharing among processes has risks, so it has to be carefully controlled. Modern OSes assume that processes that share resources trust each other by setting up appropriate security policies to control the access to shared objects, to ensure the safety of such sharing (e.g., [36]).

Unfortunately, the default access control policy of commodity OSes suffers from wrong assumptions about process trust relationship. For example, the built-in Windows security policy allows processes by the same user to share their open handles to resources, and the built-in Linux policy allows a parent process to access memory of a child process through ptrace [3]. These *default allow* policies assume a trust relationship among processes of the same user or between a parent process and a child process, which is not realistic in today's computing environments. As a result, such default allow policies can be abused by an attacker. In this paper, we present a concrete example, socket duplication attack, which enables a malicious process to impersonate a legitimate process in the interaction with an external entity over the network.

Another underlying problem that enables ShadowMove is the lack of proper message origin integrity checks in many application protocols such as FTP and TDS (for MS SQL). As a result, endpoints cannot verify the origins of the messages to ensure that the messages are not interleaved by malicious actors. An attacker who duplicated a socket can interject a request in between requests of a client and mislead the server to think the original client sent it, thus processing the request.

We can divide application protocols into three categories with regard to enforcing message origin integrity:

- **No origin integrity enforcement**. Such protocols do not have any built-in mechanisms that enable the server to check the origin integrity of the received messages, so any proper message that conforms with the protocol is accepted by the server. They are susceptible to ShadowMove attacks and one representative protocol is FTP.
- **Inadequate origin integrity enforcement**. In these protocols, the server generates a random nonce for the client to use along with its requests, and the server uses this nonce to validate the origin of received requests. Unfortunately, these protocols are not safe against ShadowMove because the attacker can wait for the client to create new connections and listen to the response from the server to learn the nonce. One representative protocol is WinRM.
- **Adequate origin integrity enforcement**. In these protocols, part of the information needed for validating origin integrity is generated by the client and not by the server. In this case, there is no way an attacker can learn that piece of information by listening to server response. These protocols are immune to ShadowMove and one representative protocol is SSL.

## 2.2 Threat Model

We assume that attackers have established a foothold on a victim system under a normal user's privilege, and they want to make a lateral movement towards the critical asset(s). The attackers have to run malware to achieve this. We assume that the victim process whose TCP connection is going to be hijacked is not aware of the malware process.

**Demonstration Scenario** We use an Employee Self-service Application of a company as an example. This is a typical multi-tier enterprise application that can be accessed from a browser. Below is the description of the components of such a system:

- Employee desktop computers, which run the web client. Some employees are IT personnel at the same time, and they need to occasionally push content to the application server, so their computers have file copying tools (such as FTP) installed.
- Application server, which runs many applications such as payroll, stock, health insurance, retirement plan, and travel.
- Database server, which stores personnel information such as DOB, SSN, contact info, and salary, and is accessed by the application server.

In this example, attackers landed on an employee desktop (via spearphishing), and this employee happens to be an IT personnel. The critical assets that the attackers go after is employee information stored on the database server. Therefore, attackers need to move from the desktop to the application

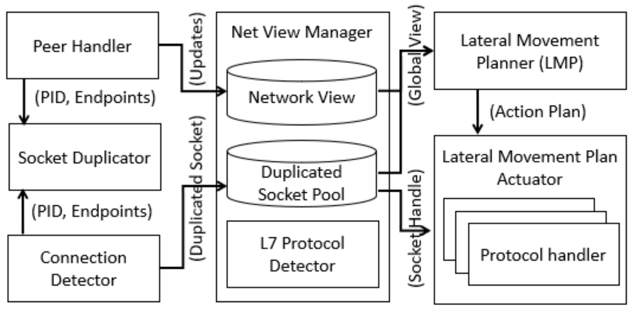


Figure 2: ShadowMove Architecture

server then to the database server. Moreover, they need to have some tool persist on the database server in order to get daily reports about updates to employee records.

To move from the desktop to the application server, the attacker can leverage the FTP connection (see Section 4.2) to copy a piece of malware to the application server and wait for the malware to be executed. For example, it is common that an application server can run an external program (e.g., data processing app implemented in C) in a path specified in a configuration file [4]. The configuration file may contain "commandname = $C:\backslash users\backslash alluser\backslash appdata\backslash updater\backslash dpanalyzer.exe$" and based on this the application server executes *dpanalyzer.exe* once some relevant event is triggered. To keep the application server up to date, an IT personnel is authorized to copy files to the application server in order to update *dpanalyzer.exe*. Under this circumstance, the attacker can leverage the FTP connection to copy a piece of malware to the application server to replace the legitimate *dpanalyzer.exe* and then wait for the malware to be executed by the application server. The attacker can get the configuration file's content via the same FTP connection.

When the malware is launched on the application server (e.g., as *dpanalyzer.exe*), it can leverage the database connection (such as Microsoft SQL discussed in Section 4.3) between the application server and the database server to copy and launch further malware on the database server.

## 3 ShadowMove Architecture and Design

Figure 2 depicts the overall architecture of ShadowMove, which consists of six major modules: Connection Detector, Socket Duplicator, Peer Handler, Network View Manager, Lateral Movement Planner, and Plan Actuator.

Central to the ShadowMove design is the notion of *Network View*, which represents a model of the normal network communication pattern in the victim environment, collectively maintained by ShadowMove instances running on different victim systems. Figure 6 gives an example network view. Each ShadowMove instance maintains two views: the *local view* is based on the current connections in the local system, and the *global view* is constructed by exchanging and propagating information among ShadowMove instances.

The Connection Detector module (Section 3.1) is responsible for detecting newly-established TCP connections that can be exploited for lateral movement and requesting the Socket Duplicator to duplicate the corresponding socket. It also detects the teardown of TCP connections and notifies the Network View Manager.

The Socket Duplicator (Section 3.4) duplicates sockets owned by target processes and passes along such sockets to its caller together with additional contextual information such as the PIDs of the owner processes.

The Peer Handler (Section 3.2) communicates with neighboring ShadowMove instances to synchronize their views of the compromised network. On one hand, it updates the Network View Manager with information learned from its peers (e.g., newly discovered hosts); on the other hand, it sends the network view of the local ShadowMove instance to its remote peers.

The Network View Manager (Section 3.3) combines a few methods to maintain a global view of the victim network, based on notifications from the Connection Detector and the Peer Handler. It also determines the service type supported by each duplicated socket and maintains the liveness of the duplicated sockets.

Periodically, the Lateral Movement Planner (Section 3.5) creates a lateral movement plan based on the current network view and the capabilities supported by the duplicated sockets. The plan specifies the socket that must be used, the type of action that must be carried out, and the payload.

Finally, the Plan Actuators (Section 3.6) execute individual steps in a lateral movement plan, such as transferring a file to the remote server, by sending packets to and/or receiving packets from the given sockets.

### 3.1 ShadowMove Connection Detector

Two approaches exist for detecting and tracking TCP connections. First, we can periodically poll TCP connection information and compare the returned information with the result of the previous call. This approach is used by tools such as TCPView on Windows. A second approach is event-driven in which we register an event handler for the creation or teardown of connections. In Windows OS, one can get information about connection state changes by creating a WMI (Windows Management Instrumentation) filter and registering a WMI event consumer [57]. However, registering a WMI event consumer requires administrative privilege.

As a result, we choose the first approach. By calling `GetTcpTable2` and `GetTcp6Table2` on Windows, or by running the command `netstat -ntp` on Linux, the Connection Detector can get basic information about TCP connections, such as connection state, local IP address, local port, remote IP address, remote port, and the ID of the owner process [42]. From the process ID it can further get the process name. When the Connection Detector observes a connection state change from non-ESTABLISHED to ESTABLISHED,

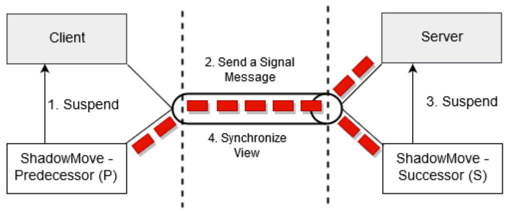


Figure 3: ShadowMove - Synchronization Signal

it invokes the Socket Duplicator about the new TCP connection and then notifies the Network View Manager to add the duplicated socket into the pool. On the other hand, when it observes a connection state change from ESTABLISHED to non-ESTABLISHED, it notifies the Network View Manager to remove a duplicated socket from the pool because the associated TCP connection becomes unusable. The notification message contains basic information of the TCP connection and the owner process name.

On Windows, the Connection Detector does some simple filtering of TCP connections before it notifies the Socket Duplicator or the Network View Manager. Specifically, it checks whether the ShadowMove process has enough permission to open the owner process of a TCP connection with PROCESS_DUP_HANDLE access flag, and it skips those connections for which the ShadowMove process does not have enough permission.

## 3.2 Peer Handler

The Peer Handler module enables ShadowMove instances to share their views of the compromised network with their neighboring ShadowMove instances. Each instance *I* uses the shared information to construct a global view of accessible systems via already-compromised systems. The Peer Handler module is executed in a separate worker thread.

Upon execution, the Peer Handler attempts to locate a configuration file in the working directory of *I*. This file contains information about the TCP connection that was used to move *I* to the current system. ShadowMove then determines the corresponding server process and the socket that were misused by the predecessor ShadowMove instance. It duplicates this socket by calling the Socket Duplicator module and then continuously listens to the incoming traffic of the duplicated socket.

As shown in Figure 3, on a regular basis, the predecessor ShadowMove suspends the client process and then sends a special request to the remote server. Upon receiving this "signal" message, the successor ShadowMove suspends the server process. Then these two ShadowMove instances can synchronize their knowledge about the network using a protocol similar to the distance vector routing protocol [56].

## 3.3 Network View Manager

This module maintains a global view of the victim network based on information received from the Connection Detector and the Peer Handler.

It manages the *Duplicated Socket Pool* and keeps a tuple <connection state, local IP address, local port, remote IP address, remote port, service type, owner PID, owner process name> for each socket in the pool. Most of these fields are passed in by the Connection Detector, except for service type (or protocol), which it determines in a sub-module called *Layer 7 Protocol Detector* by combing a few methods. First, it guesses from the destination port because many services run behind well-known default ports [11], e.g., the default port number for FTP is 21. Second, it guesses from the owner processes if they are well-known client-side tools for some services, e.g., ssms.exe or the Microsoft SQL Server Management Studio is a client of SQL server. Finally, if the port number and the owner process information are not sufficient for a reliable guess, it passively sniffs the network traffic by calling the `recv` API on each socket and setting the *MSG_PEEK* flag. Then it analyzes the received payload to recognize the application-level protocol, leveraging existing protocol analysis techniques such as automatic protocol detection feature in Suricata [55].

Based on the Duplicated Socket Pool, the Network View Manager computes a local view, which can be represented by several predicates shown in Table 2: a *system* predicate defines the IP address of a host, and a *connected* predicate defines connections between two systems. When it receives notifications from the Peer Handler, which are *system* and *connected* predicates shared by the neighbors, it updates its global view by merging the predicates into its local view.

It is worth noting that, in Windows, closing a socket does not always entail in TCP connection termination handshake. The termination handshake occurs only when the last socket descriptor is closed. As a result the connections will remain open even if owner processes close their sockets. However, a TCP connection may be not usable because of several reasons such as network failure, remote process crash, or connection inactivity timeout. To prevent connection inactivity timeout to occur, the Network View Manager sets the *SO_KEEPALIVE* flag for all duplicated sockets using `setsockopt` API function; by doing so, keep-alive packets will be sent through these connections automatically.

## 3.4 ShadowMove Socket Duplicator

The Socket Duplicator duplicates sockets associated with given TCP connections when it receives a request from the Connection Detector or the Peer Handler. The underlying idea of our approach is to duplicate the socket inside the target process and to use the resulting socket to secretly access the established TCP connection.

### 3.4.1 Socket Duplication on Windows

On Windows, one can call `DuplicateHandle` API to duplicate different types of handles from a remote process. However, as mentioned in `DuplicateHandle` documentation [40], this function cannot be used to duplicate sockets.

Although Windows offers an API named WSADuplicateSocket to duplicate a socket, we cannot directly use this function as it requires cooperation between the processes. As mentioned in [41], a typical scenario of using this function goes as follows. A source process creates a socket and wants to share it with a destination process. First, the source process calls WSADuplicateSocket to get a special WSAPROTOCOL_INFO structure. This info structure is given to the destination process via inter-process communication (IPC) mechanism. The destination process passes the info structure to WSASocket to reconstruct the socket on its side. The main challenge in this approach (i.e., using WSADuplicateSocket) is that both processes must cooperate with each other to duplicate a socket, which is not the case in our scenario where the attacker wants to duplicate a socket from an unwary victim process. One way to address this issue is to inject code into the victim process to implement the missing steps due to a lack of cooperation. However, existing defense mechanisms such as Windows Defender ATP flag usages of common process injection techniques [48], which makes the solution less attractive.

We devised a novel technique, by using Windows APIs in an unconventional way, that enables an attacker process to duplicate a socket from a target process without requiring its cooperation. Table 1 depicts the steps that the attacker process performs to duplicate a socket from a target process, assuming it knows the process ID of the target, thanks to real-time connection detection (Section 3.1). First, it opens the target process by using OpenProcess to enumerate all of the open handles in the target. The attacker process only seeks for file handles with the name of \device\afd (steps 3-5, and afd stands for ancillary function driver). During this operation, the attacker process duplicates all file handles as it is required for reading the name of a handle. We discover that the attacker process could treat these duplicated afd handles as sockets. To locate the exact socket corresponding to a TCP connection, the attacker process obtains the remote IP address and remote port to which the afd handle of socket is connected (by invoking getpeername) and compares them with the information passed in by the Connection Detector. If there is a match, the attacker process passes the afd handle to WSADuplicateSocketW to obtain the information necessary for duplication of the original socket. After obtaining the protocol info structure, the attacker process calls the WSASocketW function to duplicate the socket. This socket is then saved in the Duplicated Socket Pool together with context information such as the owner PID, the owner process name, local IP address, local port, remote IP address, and remote port.

It is also worth noting that on Windows, the TCP connection tables for IPv4/6 only contain information about the original socket descriptors not the duplicated ones and the owner PID of a socket descriptor will never change even after the termination of the owner process. This means that conventional tools such as netstat, which rely on Windows APIs to retrieve TCP connection tables, cannot be used to detect whether a connection is duplicated and nor its duplicators.

### 3.4.2 Deep Dive into Socket Duplication on Windows

To understand why ShadowMove's socket duplication works, it is necessary to first understand *socket context*. The winsock2 libraries maintain socket context for each socket handle in a number of data structures at different layers ( [58] and Figure 4). Inside WS2_32.dll, there is a hash table called sm_context_table, which maps a socket handle to a DSOCKET object that stores information about the socket such as the process and service provider. At the next layer, mswsock.dll (a service provider), there is another hash table called SockContextTable, which maps a socket handle to a SOCKET_INFORMATION object, which stores information such as socket state, reference count, local address, and remote address. Every user-level operation on the socket, such as connect, send, and recv, has to refer to and may change the socket context (e.g., the remote address and the reference count). Moreover, such context information including the hash tables is maintained for each process. The kernel side of socket functionality, which is the ancillary function driver or AFD.sys, also maintains socket context information (e.g., local address and remote address), which is necessary for the kernel driver to eventually construct network packets.

**What happens during normal socket sharing via WSADuplicateSocket**. The normal socket sharing on Windows [40] involves three steps, as illustrated in Figure 4. When the source process invokes WSASocket to create a new socket, it does three things [58]: (1) calling NtCreateFile to get a socket handle (e.g., Handle 1), (2) creating a new SOCKET_INFORMATION object for Handle 1, and (3) calling NtDeviceIoControlFile to set the kernel side context information of Handle 1. Next, when the source process invokes WSADuplicateSocket to share Handle 1 with the destination process, it first creates a duplicate of Handle 1 (e.g., Handle 2), and then puts Handle 2 in the dwProviderReserved field of a WSAPROTOCOL_INFO structure to be shared with the destination process [59]. When the destination process invokes WSASocket with the WSAPROTOCOL_INFO structure as one parameter, WSASocket extracts Handle 2 from the dwProviderReserved field and uses it to call NtDeviceIoControlFile to get the kernel side context information; once this is done, it uses the obtained information to construct an SOCKET_INFORMATION object for Handle 2, which makes Handle 2 a functional socket handle.

**What happens during ShadowMove's socket hijacking (Table 1)**. Using the same scenario above, our ShadowMove attack can secretly share the socket with handle Handle 1 without the cooperation of the source process. ShadowMove also uses a combination of WSADuplicateSocket and WSASocket, but it does one more step as preparation: it first creates a duplicate of Handle 1 by calling NtDuplicateObject; this is necessary because Handle 1

Table 1: ShadowMove Socket Duplication Given Owner Process ID, Remote IP, and Remote Port Number

| Step | Description | Kernel/ntdll Functions |
|---|---|---|
| 1 | Open the owner process with PROCESS_DUP_HANDLE | OpenProcess(PROCESS_DUP_HANDLE, , pid) |
| 2 | Foreach handle with type 0x24 (file) | NtQuerySystemInformation(SystemHandleInformation, ...) |
| 3 | Duplicate the handle | NtDuplicateObject |
| 4 | Retrieve its names | NtQueryObject(ObjectNameInformation) |
| 5 | Skip if the name is not \device\afd | |
| 6 | Obtain remote IP and remote port number | getpeername(handle, ...) |
| 7 | Skip if remote IP and port do not match the input parameters | |
| 8 | Call WSADuplicateSocketW to get a special WSAPROTOCOL_INFO structure | WSADuplicateSocketW(handle, ...) |
| 9 | Create a duplicate socket | WSASocketW(WSAPROTOCOL_INFO, ...) |
| 10 | Use the socket | recv(), send() |

is in the address space of the source process so ShadowMove cannot directly operate on it, but ShadowMove can directly use the duplicate handle (e.g., Handle 1') because it is created in the context of ShadowMove. Next, ShadowMove invokes `WSADuplicateSocket` to share Handle 1' with itself. As a result, Handle 2 is created and put in the `dwProviderReserved` field of the `WSAPROTOCOL_INFO` structure. Finally, ShadowMove invokes `WSASocket` with the `WSAPROTOCOL_INFO` structure as one parameter, in order to make Handle 2 a functional socket handle. Here since `WSADuplicateSocket` and `WSASocket` are invoked in the same process (i.e., ShadowMove), there is no need to pass `WSAPROTOCOL_INFO` structure across processes.

### 3.4.3 Socket Duplication on Linux

Our design of socket duplication on Linux (or *NIX in general) is different from its Windows counterpart. Due to a stricter process isolation, it is not possible to duplicate a socket from another process directly, even if the other process is owned by the same user. However, socket sharing is supported on Linux, but it requires cooperation between the two processes. Since ShadowMove assumes that the victim application is not cooperative, our solution is to force the victim application to cooperate by injecting code into its address space to set up the sharing of a socket with the ShadowMove process. To inject code into the victim application, we create a launcher that would run the victim application as a child process and then leverage `ptrace` to inject code, in the form of a shared library. Finally, we put the launcher version ahead of the original victim application in the command search path, such that the user would invoke our launcher when he/she intends to run the victim application.

We should note that the use of process injection can jeopardize the stealthiness of the ShadowMove attack on Linux, compared with ShadowMove on Windows. However, our Linux design still has a good chance of evading state-of-the-art defenses. We defer a detailed discussion to the evaluation (Section 5).

**Socket sharing on Linux.** To share a socket, two processes first connect via a Unix domain socket, then the sender process invokes `sendmsg` and passes the socket descriptor in the input parameter, while the receiver invokes `recvmsg` and retrieves a (possibly different) socket descriptor from the output parameter. When a socket descriptor is passed this way, the underlying Linux kernel creates a new descriptor in the receiving process' address space that refers to the same file table entry within the kernel as the descriptor that was sent by the

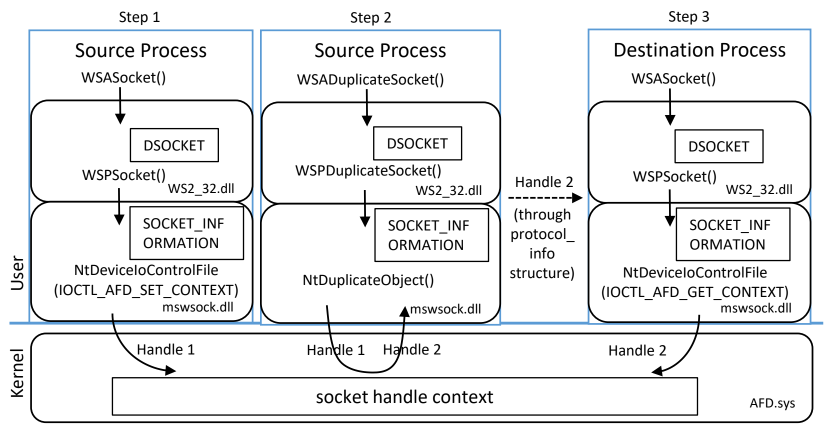


Figure 4: Winsock Duplication

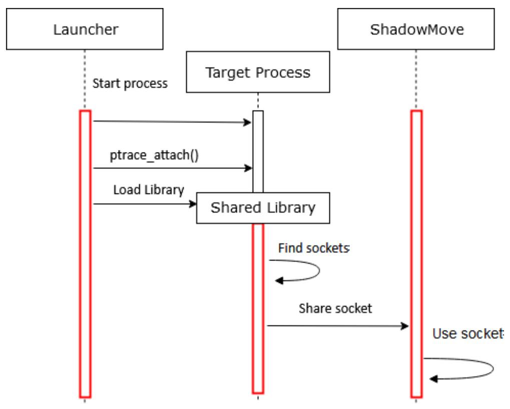


Figure 5: ShadowMove in Linux System

sending process [54].

More specifically, there are four components for a ShadowMove attack on Linux, which are target process, shared library, launcher, and ShadowMove (Figure 5).

The launcher injects a shared library into the target process by using `ptrace` [32, 47], which has to attach to the target process first. The current Linux systems impose strict control over `ptrace`. Specifically, by default the Yama Linux Security Module (LSM) [3] only allows `ptrace` from a process with sudo privilege, or from a parent to a child process. We use the second option because for this we don't need privilege escalation. Therefore, our launcher runs the target application as a child process and then attaches to the target process using `ptrace`. After that, it invokes `__libc_dlopen_mode` for loading the shared library into the target process. Our launcher is based on an open source project [30].

We developed a prototype of the shared library, whose constructor function (executed automatically when the library is loaded) enumerates open sockets in the target process. For each open socket, it makes a copy of that socket using `dup` method, connects to the ShadowMove process through a Unix domain socket, and shares the duplicated socket using that channel. If there is no open socket, it sleeps for a while and tries to find open sockets again. To avoid blocking the main thread of the target process, we create a new thread that is dedicated to socket duplication.

To make the victim user run our launcher inadvertently when he/she intends to run the target application, we give the launcher the same name as the target application and we ensure that our launcher is ahead of the target application in the command search path, which can be done by changing the PATH environment variable. To make the attack stealthier, we can avoid changing the PATH environment variable if any location on the current command search path is (1) writable by the victim user and (2) before the location of the target application: in that case we just need to copy our launcher in that writable location. Otherwise, we would create a folder that appears benign (e.g., `/home/alice/.npm-packages/bin` that can be used by a benign application called `npm` [8]), copy our launcher there, and add the new folder location to the PATH environment variable by adding `export PATH=/path/of/the/launcher:$PATH` into the victim user's .bashrc.

For example, if `ftp` is the target application then the launcher will be named `ftp`. When the user tries to run FTP, the launcher will be executed and it will run the original FTP application as a child process.

### 3.4.4 The Race Between the Benign Application and the Attack

We should note that in the proposed attack, the socket is shared between the original client and the attacker, which can cause a race condition in receiving and sending data from the remote endpoint. The one who calls the `recv` function first will get the data from the input buffer and the one who calls `send` function first will send the data to the server. This may result in reading partial responses from the server or sending a garbled request to the server. To prevent such a possibility, attackers can simply pause the client process temporarily while they are sending/receiving data from the server and resume the client process afterwards. To suspend the client process, the attacker can pause all its threads by calling `SuspendThread`, and to resume the client process, the attacker can resume all its threads using `ResumeThread`.

Table 2: ShadowMove Predicates to Model Target Networks

| Predicate | Definition |
|---|---|
| system | system(ip_addr) |
| connected | connected(src_ip, dst_ip, service) |
| committed | comitted(src_ip, dst_ip, action) |
| capability | capability(service, action). |

## 3.5 Lateral Movement Planner (LMP)

The Lateral Movement Planner (LMP) can empower the adversary to coordinate attack actions on multiple victim systems that can optimize the attack effectiveness and stealthiness. For example, suppose the attacker in Figure 6 has compromised hosts A and B, which both connect to host C, but their individual connections are not sufficient for a lateral movement (e.g., A's connection can only copy malware, and B's connection can only execute malware). In this case, a coordinated plan that involves both A and B (e.g., A copies malware to C, then B remotely launches malware on C) would allow a lateral movement to C, thus making the attack more effective. For another example, if there exist multiple paths to the target system, a coordinated plan would allow the attacker to use the shortest path to send payload to / receive data from the target, thus making the attack stealthier. We assume that attackers look for a specific set of targets that can be recognized when they are reached.

We formulate the attack planning problem in Prolog. We uses the predicates in Table 2 to specify the current state of the compromised network: *system* and *connected* specify the reachable systems and their interconnections, and *committed* defines the action that has been performed on a system by a ShadowMove instance. For each protocol, we also use the *capability* predicate to specify the actions that attackers can do if they hijack the corresponding TCP connection.

Figure 6 illustrates an snapshot of system B's (with IP address 10.10.10.50) ShadowMove knowledge base, which consists of a set of *facts* that represent a network with three compromised systems and one target. This knowledge base is constructed from the global view shared among all ShadowMove instances. LMP uses the following rules to determine whether a specific operation can be carried out on a remote system Y from a given system X.

```
remoteOperation( X, Y, Action , Route):-
```

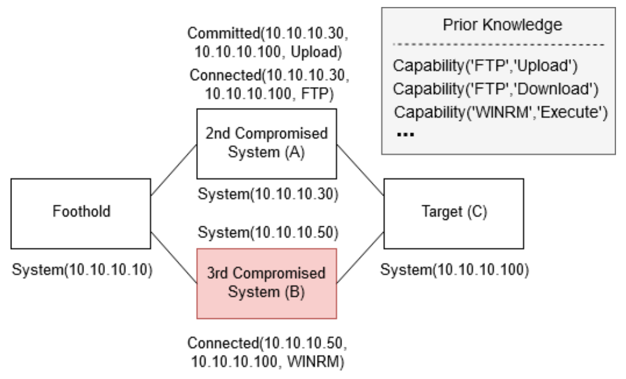

Committed(10.10.10.30, 10.10.10.100, Upload)
Connected(10.10.10.30, 10.10.10.100, FTP)
Prior Knowledge
Capability('FTP','Upload')
Capability('FTP','Download')
Capability('WINRM','Execute')
...
2nd Compromised System (A)
System(10.10.10.30)
Foothold
System(10.10.10.10)
System(10.10.10.50)
3rd Compromised System (B)
Target (C)
System(10.10.10.100)
Connected(10.10.10.50, 10.10.10.100, WINRM)

Figure 6: Example ShadowMove Network View and Knowledge Base

```
    connected(X,Y,S), capability(S, Action),
    Route=[X|[Y]].

remoteOperation( X, Y, Action, Route):-
    connected(X,Z,Service),
    capability(Service, Action),
    remoteOperation( Z, Y, Action, R),
    Route=[X| R].
```

By using *remoteOperation*, a ShadowMove instance can check whether there exists a path between two systems that would allow them to perform a specific operation such as execute or upload a file. For example, the attacker can execute the following query:

```
remoteOperation('10.10.10.10', '10.10.10.100',
    'upload', R).
```

which returns $['10.10.10.10','10.10.10.30','10.10.10.100']$. This result means that an attacker who landed on 10.10.10.10 and has moved to 10.10.10.30 can copy malware from 10.10.10.30 to 10.10.10.100 via one of the ShadowMove actuators.

We can use *remoteOperation* predicate to construct more complex predicates such as *commitExecuteOperation*:

```
commitExecuteOperation(X, Y) :-
      connected(X, Y, Z),
      capability(Z, execute),origin(I),
      remoteOperation(I, Y, upload, _R),
      committed(_K, Y, upload).
```

In order to run ShadowMove on a target system from a compromised system, not only there must be a connection between these two systems that allows the ShadowMove instance to perform execute operation, but the file must has also been uploaded to that target system by one of the ShadowMove instances prior to the execute operation. For example in Figure 6, system B can launch ShadowMove on system C (target) if and only if (1) there is a connection that allows system B to execute a file on system C:

```
      connected(SystemB, SystemC, Z),
      capability(Z, execute)
```

and (2) the ShadowMove binary file has been uploaded on system C:

```
      origin(I),
      remoteOperation(I, systemC, upload, _R),
      committed(_K, systemC, upload).
```

If based on its current knowledge base, no ShadowMove instance has uploaded the file on the target, then system B must wait until the upload operation is committed by one of the ShadowMove instances, such as the one on system A. To obtain a list of target systems that system B can launch ShadowMove on, the ShadowMove instance on system B can execute the following query:

```
findall(Target,
      commitExecuteOperation('10.10.10.50', Target),
      ExecuteList).
```

If the returned ExecuteList is not empty (e.g., ['10.10.10.100']), an instance of ShadowMove can be started on a new target system (e.g., 10.10.10.100). This is an illustration of lateral movement that requires coordination among different paths, which is only possible when a global view of the compromised network is available.

## 3.6 Lateral Movement Actuator

Lateral Movement Actuator (LMA) is a module manager containing several actuation modules. Each of these modules is responsible for handling one protocol such as TDS (Section 4.3). LMA can act both passively and actively. In the passive mode, the module only reads from a socket by passing *MSG_PEEK* flag to `recv` API call. In this way, the input buffer is not emptied, so the original process can read the content. In the active mode, the module reads from the socket without passing the *MSG_PEEK* flag; hence the `recv` call consumes the data in the input buffer. In this state, the module also writes to the socket out buffer to send crafted messages.

In some protocols, we need to learn a few secrets before being able to craft valid messages (e.g., shellID for WinRM in Section 4.4). In these scenarios, an actuator module starts in the passive mode, sniffing the receiving messages to learn such secret values. After learning all of such required data elements, the actuator module can switch itself to active mode and start communicating with the remote endpoint. It is worth noting that LMA module can only read incoming messages; it cannot read the outgoing messages as to the best of our knowledge there is no such API that allows one to read from the socket output buffer. In our current prototype, LMA has three actuation modules for FTP, MS SQL, and WinRM protocols. However, one can add a new protocol to LMA by implementing an interface called IPModule.

## 4 Prototypes for ShadowMove Actuators

We implement a prototype of the ShadowMove design on Windows in 2,501 lines of C/C++ code. The lateral movement planner is based on SWI-Prolog [14], a free implementation of the programming language Prolog. The prototype [16] showcases common functionalities such as connection detection, socket duplication, network view synchronization, and lateral movement planning; it also overcomes the challenges of actuation, i.e., how to make the injected packets conformant to the respective protocols and yet useful for lateral movement (such as uploading malware and launching malware), which is specific to individual application protocols.

In this section, we present three ShadowMove actuators that leverage FTP, MS SQL, and WinRM. The criteria for choosing these protocols is their lack of support for message origin integrity, as we discuss in Section 2.1. Specifically, FTP and Microsoft SQL have no origin integrity enforcement, and WinRM has inadequate origin integrity enforcement.

### 4.1 ShadowMove Instantiation

For each experiment, we first prepare a target environment that includes the victim applications, such as one machine running a FTP client and another machine running a FTP server. We configure the applications so that they run normally with their intended purposes. We launch ShadowMove PoC in the victim client machine. We observe that the PoC periodically detects candidate TCP connections to abuse once they are established (the victim client application does not have to start before the PoC), duplicates the corresponding sockets, and determines the protocol running over the TCP connections (e.g., FTP). The PoC periodically queries the lateral movement planner module (by presenting its current network view) and executes the actuator logic if the planner returns the target of the next move (e.g., using the FTP connection to copy the PoC to the FTP server). When the PoC is started on the server machine, we see that it detects active TCP connections (including the one with the client machine) and duplicates the corresponding sockets. We further observe that the PoC on the server exchanges "signal" messages with the PoC on the client successfully, and then they exchange their current network views. Upon doing that, the network views on both machines are updated. Some time later, lateral movement planner module is queried again to make the next decision based on the new network view.

The scenario described above is common to all three actuators presented in the rest of Section 4. Therefore, we omit such details in the description of individual actuators. A demo video of our ShadowMove PoC that leverages FTP and showcases the above scenario can be found at [16]. In this demo, we start ShadowMove PoC manually after it moves to the FTP server, but we can automatically start the PoC via WinRM, as demonstrated in Section 4.4.

### 4.2 FTPShadowMove: Hijacking FTP Sessions

We develop prototype systems that can hijack established FTP connections on Windows 10 and Ubuntu 18.04. They work under the default installation of `ftp` and do not require any elevated privileges. They allow an attacker to download and upload files to a remote FTP server without authentication.

In the FTP protocol, a client uses one TCP connection to send commands to a server and receive the corresponding responses from the server; this connection is called *command channel*. The client also uses another TCP connection to send or receive data such as file contents; this connection is called *data channel*. A client can open multiple data channels for a given command channel. Authentication is required only for establishing the command channel, which means a client does not need to re-authenticate itself for creating a new data channel. Attackers who have hijacked the command channel can send a request to the server to open a new data channel for themselves, thus avoiding any collision with the client contents that are being transferred on existing data channels. However, attackers still should adopt a strategy to prevent a race condition in the shared command channel. Note that one cannot detect the attack simply by monitoring the creation of new data channels because the legitimate client may open new data channels as well.

A FTP client can request for creating a new data channel in two ways: active FTP and passive FTP. In the active FTP, the client sends `Port` command to the server specifying the port that server needs to connect back to establish the connection. In the passive FTP, the client send `PASV` command to server, asking the server to listen to a port that client can connect in order to create a new data channel. In a nutshell, the difference between these two modes is with respect to who initiates the new TCP connection: server in active mode, and client in passive mode are supposed to connect to the port specified by client and server, respectively. In our prototype, we implemented the passive FTP for demonstration. However, active FTP can also be implemented with negligible effort.

In passive FTP, the client sends `PASV` command to the server, and the server responds back by giving the information about the endpoint, including IP address and port, that the client must connect to in order to create a new data channel. The `PASV` is documented in RFC-959.

**Experiment Setup** We deployed a vsftpd server on a Linux-based virtual private server hosted on the Internet. For the legitimate client, we used the `ftp` command and `Windows Explorer` to connect to the configured server. The anonymous login is blocked on the server so the client needs to send a valid username and password to connect to it. As can be seen in our demo video at [6] and the top half of Figure 7, the client exchanges several messages with the server in order to login to the server. After that, we launch FTPShadowMove under the same user account as the `ftp` client.

Our FTPShadowMove PoC first hijacks the FTP connection by duplicating the corresponding socket, and then it sends several commands to upload a binary file to a specific directory on the server. The specific commands (such as `CWD /files/`) and the server responses are shown in the bottom half of Figure 7. Specifically, we can see that the server responded to the

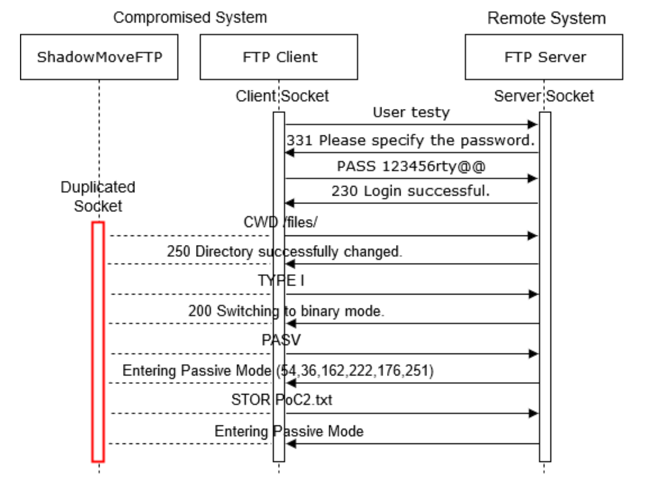


Figure 7: ShadowMove Injects Commands to Duplicated FTP Socket in Order to Open a New Data Channel Connection

PASV request and asked FTPShadowMove to connect back to 54.36.162.222 on port 45307 (i.e., $176*256+251$). FTPShadowMove then requests to upload a file named *PoC*2.*txt* on the server. After receiving response code 150 from the server, FTPShadowMove opened a TCP connection to the specified remote endpoint and sent the content of the file to the opened connection. The server interpreted the file as binary content and stored it in $/files/PoC2.txt$ on the server.

Our prototype on Ubuntu 18.04 uses the same FTP commands mentioned above, and a video clip of how it works can be found at [15].

In our prototype systems, we only used a few FTP commands. However, there are many other FTP commands that can be utilized by attackers. A complete list of all possible FTP commands can be found at [5]. Specifically, The FTP SITE command allows a user to execute a limited number of commands via the FTP server on the host machine [53]. No further authentication is required to execute the command. The commands that may be executed vary from system to system, and some useful ones include EXEC and CHMOD. The EXEC command executes provided executable on the server, which can be used to start malware. Fortunately, on many systems the SITE command is not implemented, and it is also recommended that the SITE command be disabled on FTP servers if possible.

### 4.3 SQLShadowMove: Hijacking Microsoft SQL Sessions

We have confirmed that it is possible to (1) hijack Microsoft SQL connections to upload malware executables from a SQL client machine to a SQL server, and (2) execute the malware on the SQL server.

**Experiment Setup**. We use Microsoft SQL Server Management Studio 17 as the legitimate SQL client, and Microsoft SQL Server version 14.0.1000.169 as the server. We configure a user on the SQL server who can create databases and tables. We first launch the SQL client and login to the server. Then we run our proof-of-concept SQLShadowMove. We confirm that our proof-of-concept works under the default installation of Microsoft SQL and normal application settings.

Our SQL hijacking scheme requires several preconditions to work successfully: (1) the traffic is not encrypted, (2) there is a folder on the SQL server writable by the SQL server process, (3) the SQL client has successfully authenticated to the SQL server, and (4) the SQL client assumes a role that is allowed to create a table on the SQL server.

The above preconditions can often be satisfied. By default the Microsoft SQL traffic is not encrypted, and the %TEMP% folder is always writable by any process on the SQL server [33]. Moreover, the SQL server is almost stateless. The client and the server uses the TDS (Tabular Data Stream) Protocol [44] to communicate. Although several fields in the TDS header are designed for maintaining some states, they are optional or are not used by the current implementation. For example, the SPID field in the TDS packet header is the process ID on the server corresponding to the current connection. If this ID is strictly checked, the attacker has to somehow learn it before fabricating a rogue packet. Unfortunately, this field is not required, and a value of 0x0000 is acceptable by the server. Similarly, two more fields are defined but ignored: PacketID and Window.

There are several types of TDS packets. The most relevant type to our attack is the Batch Client Request type [45], whose payload can be a Unicode encoding of any SQL statement, and there is no checksum in the packet header. This makes it straightforward to capture a real Batch Client Request packet and then use it as a template to create new rogue requests by replacing the payload with new Unicode strings; in our case, such strings correspond to a series of SQL statements.

SQLShadowMove first detects a TCP connection created by the SQL client process and duplicates the corresponding socket. Then it uses the duplicated socket to send a series of Batch Client Request packets to the SQL server, and receives any response packets from the server. The payload of these Batch Client Request packets consists of SQL scripts that upload an executable file to the SQL server and execute it.

Specifically, the SQL scripts first create a table on the SQL server, then they insert chunks of bytes from the executable file into the table. Finally, they invoke the bcp command to export content of the table to a regular file on the server, thus restoring the original executable file. The pseudo code of the SQL scripts is shown in Figure 8.

With the executable on the SQL server, our prototype can further run it through a SQL statement.

To experimentally confirm the feasibility of SQLShadowMove, we develop a simple Windows application (named notepad.exe) to represent a piece of "malware". This application creates a file (named notepad.txt) in the same folder as the application executable and writes the current date and time into that file. Then we generate SQL scripts to upload the sim-

```
• Create a table "BlobFileTable" that has a column with
  name "FileContent" and type varbinary(max).

• Add chunks of bytes from the source file into the table, e.g.,
     insert into BlobFileTable(FileContent) values (0x4D5A000000)
     insert into BlobFileTable(FileContent) values (0x89EB003401)
     More insert statements ...

• Invoke the "bcp" command to dump the content of BlobFileTable
  to a file on the SQL server

• Use xp_cmdshell to execute the dropped file.
```

Figure 8: SQL Scripts Used by SQLShadowMove

ple "malware" to *%TEMP%\notepad.exe* on the SQL server and run it. After we run the proof-of-concept of SQLShadowMove, we can visually confirm that first *notepad.exe* appears on the SQL server, and then *notepad.txt* appears and its content matches the time and date on the SQL server. A video clip of how SQLShadowMove works is available at [17].

Note that in order to run the `bcp` command or the executable file, `xp_cmdshell` has to be enabled on the SQL server. However, this is not a hurdle for our prototype because our SQL scripts enable `xp_cmdshell` before using it.

## 4.4 WinRMShadowMove: Remote Execution Based on WinRM

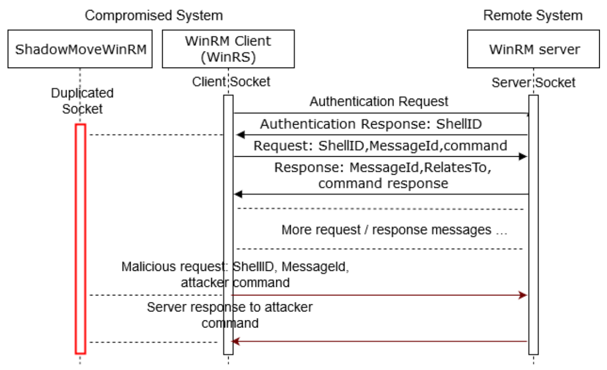
Windows Remote Management (WinRM) is a feature of Windows that allows administrators to remotely run management scripts [39]. We have confirmed that it is possible to hijack WinRM sessions to run malware on a remote machine. We assume that the remote machine is running the WinRM service and the malware has been uploaded to the remote machine and it just needs to be launched.

### 4.4.1 Brief Introduction to the WinRM protocol

WinRM protocol [39, 49, 60] uses HTTP to communicate with the remote server. To authenticate with remote machine WinRM has six authentication mechanisms: Basic, Digest, Kerberos, Negotiate, Certificate and CredSSP. By default, it uses Negotiate. A WinRM client first authenticates with the WinRM server. After authentication the WinRM client receives a shellID from the server, which is used in later communication. Besides shellID there are a few other IDs in every request message. The messageID is used to pair a response message with the corresponding request message, and in the response message, the request messageID is present as the "RelatesTo" field. Figure 9 illustrate the message exchanges during a WinRM session.

### 4.4.2 Experiment Setup

To prepare the environment for WinRM hijacking, we first set up WinRM for a normal application scenario on Windows 10, which includes enabling WinRM on both the server and the client, and adding the server as a trusted host on the client machine. Then we can use the commandline tool `winrs` on the client machine to run commands on the server.



Compromised System
Remote System
ShadowMoveWinRM
WinRM Client (WinRS)
WinRM server
Client Socket
Server Socket
Duplicated Socket
Authentication Request
Authentication Response: ShellID
Request: ShellID,MessageId,command
Response: MessageId,RelatesTo, command response
More request / response messages ...
Malicious request: ShellID, MessageId, attacker command
Server response to attacker command

Figure 9: ShadowMove Injects Attack Payload to Execute a Binary in the Remote System.

However, ShadowMove does not work under the above default setting because WinRM traffic is encrypted by default. In order for our WinRMShadowMove PoC to work, an administrator has to configure the WinRM server to allow basic authentication and to allow transfer of unencrypted data. We should note that this kind of configuration is not rare because it can get WinRM to work quickly, and some third party WinRM client and libraries [1] require unencrypted payload to communicate with the WinRM server. We use this configuration in our experiement, and more details of the configuration can be found in the Appendix (Section A).

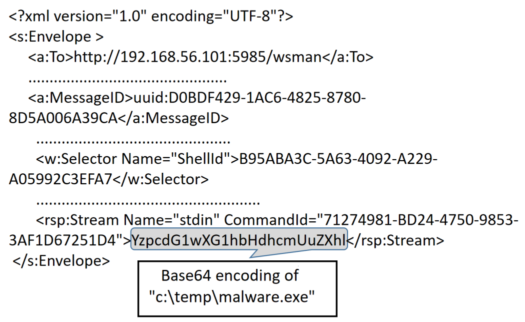
### 4.4.3 Hijacking WinRM

To demonstrate how WinRMShadowMove works, on the client machine, we run the commandline `winrs -un -r:http://host_ip:5985 -u:user -p:pass cmd`, which will create a new `winrs` process and open a command shell to the remote machine. The `-un` flag specifies that the request and response messages will not be encrypted. Concurrently in another terminal, we run WinRMShadowMove.

As the `winrs` process starts execution, it establishes a TCP connection to the WinRM server, which is captured by the Connection Detector. As a result, the Connection Detector notifies the Socket Duplicator, which finds and duplicates the socket inside the `winrs` process. WinRMShadowMove first runs in the passive mode (i.e., peeking into the incoming network packets through the duplicated socket) in order to learn the shellID from the server; then it switches to the active mode. Here we use the idea discussed in Section 3.6.

Because the WinRM server supports unencrypted payload, we can construct a plain text HTTP payload and send it to the server through the TCP socket. For this scheme to work, the constructed payload must appear legitimate to the server. After analyzing the HTTP request and response packets using Wireshark, we found that MessageID is unique for every payload and it is actually a UUID. Therefore, we use a UUID generator to generate messageID. Furthermore, we get the shellID from the authentication response message. Using these two IDs we can construct a payload to execute an executable file on the remote WinRM server.

To learn how to construct the payload, we leveraged an open source WinRM client called winrm4j [2] to communicate with a remote WinRM server, and we use the request packets generated by winrm4j as the template for our payload. Figure 10 shows the payload of an example WinRM request.



```
<?xml version="1.0" encoding="UTF-8"?>
<s:Envelope >
   <a:To>http://192.168.56.101:5985/wsman</a:To>
   ..............................................
   <a:MessageID>uuid:D0BDF429-1AC6-4825-8780-
8D5A006A39CA</a:MessageID>
     ..............................................
     <w:Selector Name="ShellId">B95ABA3C-5A63-4092-A229-
A05992C3EFA7</w:Selector>
     ...................................................
     <rsp:Stream Name="stdin" CommandId="71274981-BD24-4750-9853-
3AF1D67251D4">YzpcdG1wXG1hbHdhcmUuZXhl</rsp:Stream>
 </s:Envelope>
```

Figure 10: A WinRM Request for Running malware.exe on a WinRM Server Whose IP Address is 192.168.56.101

Before sending the payload to remote machine using the hijacked TCP socket, WinRMShadowMove suspends the legitimate process to prevent it from getting the response message from the WinRM server. After getting the response from the WinRM server it resumes the legitimate client. The time interval between the suspension and resumption is very short, so the legitimate client may not notice it.

Figure 9 shows the interleaving of the attack messages with the legitimate WinRM messages.

## 5 Evaluation of ShadowMove Proof-of-concepts

### 5.1 Theoretical Evaluation

As we demonstrate in Section 5.2, ShadowMove cannot be detected by the current state-of-the-art lateral movement detectors. In this section, we discuss the underlying reasons that make such existing solutions ineffective in the detection of ShadowMove lateral movements.

At the host level, to perform lateral movements, our design of ShadowMove on Windows relies on a few API functions that are also commonly used by other benign processes. For example, as mentioned in [18], many processes on Windows call `OpenProcess` with `PROCESS_ALL_ACCESS` access flag, which is essentially asking for all possible permissions on the target process, including permission for duplicating its handles. Moreover, ShadowMove calls `WSADuplicateSocket` that also has legitimate use cases such as offloading sockets to child processes. Second, it is hard to trace back from a socket descriptor to all processes that have access to it, because only the process ID of the owner is recorded in a socket descriptor.

Our current design of ShadowMove on Linux requires stronger assumptions about the attacker because it relies on process injection to force victim applications to cooperate, which makes it less stealthy than its Windows counterpart (e.g., by monitoring the runtime integrity of the code sections of benign applications, one can detect the effect of code injection [31]). Moreover, since our design may modify configuration of the system (e.g., the PATH environment variable and .bashrc), one could detect it by monitoring such changes. However, despite these constraints, ShadowMove on Linux is still a viable attack.

Specifically, there are practical challenges to detect ShadowMove attacks on Linux. To the best of our knowledge, *runtime* code integrity monitoring for applications are not supported in current Linux distributions, and known monitoring tools require a hypervisor (e.g., [31]) or special hardware (e.g. [61]). Monitoring configuration changes to detect ShadowMove is also non-trivial because many benign applications (such as `npm` [8]) also make changes to both the PATH environment variable and .bashrc; a monitoring tool thus has to check precise conditions (most likely application specific) in order to avoid false alarms. As we mention in Section 3.4.3, we hide our launcher under seemingly benign paths (such as `/home/alice/.npm-packages/bin`), which further raises the bar for detection. This is corroborated by our experience with several popular host-based IDSes on Linux today: OSSEC [10], Osquery [12], and Wazuh [7], which fail to detect ShadowMove using their existing rules. Of course, one can add new rules to detect specific instances of ShadowMove, but the effort will be non-trivial.

At the network level, ShadowMove tunnels its messages through existing connections established by benign processes on both ends. In other words, it injects its messages within the streams of benign messages send by a benign client to a remote service. Hence, anomaly-based solutions that detect unusual new connections are oblivious to ShadowMove. Moreover, ShadowMove begins the lateral movements after the required authentication steps are performed by the client and the remote server. This means that ShadowMove operations do not entail any additional authentication attempts. As a result, those anomaly detection solutions that correlate user login activities with network connection activities such as [51] are ineffective.

### 5.2 Experimental Evaluation

In this section, we extensively evaluate ShadowMove in the presence of host and network-based defensive mechanisms that are typically found in enterprise environments. To be more specific, we test ShadowMove against emerging Endpoint Detection and Response (EDR) systems, top-notch antivirus products, host-based IDSes, and network-based IDSes.

We evaluate ShadowMove in the presence of emerging Endpoint Detection and Response (EDR) systems, namely CrowdStrike Falcon Prevent and Cisco AMP. EDRs are relevant to our evaluation because some EDRs (such as CrowdStrike Falcon [34]) are designed to detect lateral movements.

Table 3: Effectiveness of Antivirus, IDS, and EDR Products against FTPShadowMove (F), SQLShadowMove (S), and WinRMShadowMove (W) PoCs. N means "not detected" and – means "not applicable".

| Type | Name/Version | Update | F/S/W |
|---|---|---|---|
| AV | McAfee/16.0 | 2/3/2019 | N/N/N |
| AV | Norton/22.16.2.22 | 2/3/2019 | N/N/N |
| AV | Webroot/9.0.24.37 | 2/3/2019 | N/N/N |
| AV | Bitdefender/6.6.7.106 | 2/3/2019 | N/N/N |
| AV | Windows Defender/4.18.1901.7 | 2/3/2019 | N/N/N |
| NIDS | Snort/2.9.12 (Windows and Linux) | 2/7/2019 | N/N/N |
| HIDS | OSSEC/3.4.0 (Linux) | 10/12/2019 | N/–/– |
| HIDS | Osquery/4.0.2 (Linux) | 10/24/2019 | N/–/– |
| HIDS | Wazuh/3.10.2 (Linux) | 10/24/2019 | N/–/– |
| EDR | Cisco AMP/6.1.5.10729 | 6/14/2018 | N/N/N |
| EDR | CrowdStrike Falcon Prevent/4.20.8305.0 | 2/11/2019 | N/N/N |

We also evaluate ShadowMove in presence of host-based antivirus products: we choose the top four antivirus products ranked by [50] for our evaluation (McAfee, Norton, Webroot, and Bitdefender); we also choose Windows Defender because it is the default AV on Windows systems. Moreover, we choose the Snort IDS to evaluate ShadowMove against network-based solutions (Snort rules V2.9.12 is used). Finally, for our ShadowMove design on Linux, we use three popular host-based IDSes (OSSEC [10], Osquery [12], and Wazuh [7]) to evaluate it.

**Stealthiness against EDR and IDS solutions**. We experimentally confirmed that ShadowMove PoCs can evade the detection of Strike Falcon Prevent, Cisco AMP, OSSEC, Osquery, Wazuh, and Snort (Windows and Linux). The detailed result is shown in Table 3. During the evaluation, we used the default detection rules provided by such tools. We also manually inspect these default rules to understand why they cannot detect ShadowMove. For example, the default Osquery rules do not mention ptrace or process injection at all.

**Stealthiness against host-based antivirus products**. We also experimentally confirmed that ShadowMove PoCs can evade the detection of the latest version of the above five AVs on Windows 10 (These AVs do not have Linux versions). The overall result is shown in Table 3.

**Vendor feedback**. We contacted Microsoft Security Response Center (MSRC) and a case (number 46036) was opened for our reported issue. On June 21, 2018, MSRC dismissed our reported issue as a vulnerability, stating that "this behavior is by-design ... because from a system security standpoint, one cannot duplicate a handle from a process without already having full control over it and at that point there are many other attacks possible." This feedback from Microsoft engineering team confirmed that our attack is non-trivial to deal with because fully addressing it will require a re-design of the access control mechanism of handles in Windows. This also implies that techniques like ShadowMove will continue to help attackers on Windows in the foreseeable future.

## 6 Discussions and Future Work

**Possible mitigation of ShadowMove**. ShadowMove attacks can be mitigated by addressing the two fundamental weaknesses in existing computing environments (Section 2.1). One idea is to better isolate legitimate processes from potential attacker processes to prevent socket stealing. For example, we can make the legitimate processes as Protected (introduced in Vista) or Protected Process Light (introduced in Windows 8.1) processes, such that an unprotected process cannot open legitimate processes with `PROCESS_DUP_HANDLE`. However, this approach has limitations such as processes that have GUI cannot be protected [21] and the program file must be signed by Microsoft [27]. Another idea is to introduce strong origin integrity mechanisms in common enterprise computing protocols, like what SSL does. However, this may break many legacy applications.

**Limitations of the current ShadowMove prototype**. First, it has to find an unencrypted TCP channel because it is a user-level attack that cannot obtain secrets inside the victim process. Due to this limitation, ShadowMove cannot hijack connections for which user-level encryption is applied to the payload. One known way to hijack encrypted connections is to inject code into victim processes, which will be able to access plaintext messages [19]. Unfortunately, process injection would make ShadowMove more visible to existing detection tools (e.g., Windows Defender ATP [48]). Besides, presence of encryption may not always be a hurdle for ShadowMove: there are proposals to implement encryption service (such as TLS) in the kernel space [46], which will make the TLS session vulnerable to ShadowMove because unencrypted payload is sent to or received from the socket interface in systems that deploy such kernel-level services. Second, ShadowmMove may not be able to get information such as the shellID in Section 4.4 from the receiving buffer if the legitimate client consumes the buffer first. However, attackers can simply retry and they need to succeed only once to achieve lateral movement. Third, our design of ShadowMove on Linux injects code into the target process' address space in order to hijack its control flow, which jeopardizes ShadowMove's stealthiness compared with its Windows counterpart.

**Other attacks enabled by socket duplication**. As discovered by Bui *et al.* [20], TCP communication among applications inside a machine (such as a browser and a backend password manager) is not totally secured. Therefore, our socket duplication technique can be used to intercept and steal sensitive data from such applications. Moreover, in this study we try to abuse mostly client-side sockets (although we also abuse server-side sockets to synchronize the network view, as described in Section 3.2). However, we can use the same technique to exploit server applications. For example, by duplicating sockets used by a server application, we can inject malicious data to mount a phishing attack against a client machine, hence providing an alternative implementation for the attack described in [23].

## 7 Related Work

Traditionally, attackers exploit vulnerabilities in network services, such as SMB or RDP, to laterally move across networks. However, due to the advances in defense mechanisms, finding such vulnerabilities and exploiting them successfully without being detected has become increasingly hard. As a result, attackers have shifted their attention to more fruitful approaches such as harvesting credentials from compromised systems and reusing them to do the lateral movement. In credential dumping approach [43], attackers retrieve plaintext account information including passwords from memory of processes such as LSASS. Several open source frameworks such as Mimikatz exist that can carve passwords from various locations in a system. Similarly, attacker can leverage SSH Agent Forwarding [29] for lateral movement, in which the attacker reuses saved SSH private keys in the memory to log into SSH server(s). However, this technique requires a number of special conditions, such as client and server(s) are configured to use public/private key pairs, the client runs a SSH key agent, the victim user has added private keys to the key agent, and the attacker knows the usernames associated with the private keys. Instead of retrieving the credentials, it is also possible to harvest and reuse security tokens, such as Kerberos TGT, Kerberos service ticket, and NTLM hash, to get access to other systems in a network. Many APT groups, including APT 19 and ATP 32, use such techniques to expand their access across the target networks.

Several approaches aim to detect credential reuse attacks. Siadati *et al.* [51] propose a machine learning framework that extracts normal users' login patterns and identifies login attempts that deviate from such patterns as attacks that try to reuse learned credentials in a greedy way (i.e., testing all credentials on all reachable systems). Kent et al. [35] suggest that user authentication graphs be used to detect credential misuse in large-scale, enterprise networks.

The hijacking approach presented in this paper is different from traditional hijacking such as session hijacking in web applications and network-level TCP hijacking. Instead, what we propose is a host-level TCP hijacking by performing socket duplication. SSH-Jack [19] is a technique that injects code into the memory of a legitimate SSH client in order to establish a rogue SSH session via the SSH client, which is trusted by the SSH server. Unlike SSH-Jack, ShadowMove is application-agnostic in the sense that it does not need to know the internal implementation of clients in order to inject commands. ShadowMove is also protocol-agnostic and can be extended to support other protocols. In the current prototype, ShadowMove can handle FTP, WinRM, and TDS protocols.

SSH connection persistence (with options such as `ControlMaster`, `ControlPath`, and `ControlPersist`) [9, 13] is a SSH feature that can be abused for lateral movement. With SSH connection persistence, a *master* SSH client process goes through the normal authentication steps to establish a connection to a SSH server; then *slave* SSH clients can reuse this connection to access the server without repeating the authentication steps. Therefore, if the victim environment has a master SSH client running, an attacker can make a lateral movement to the SSH server by acting as a slave SSH client. However, this attack requires process cooperation: a SSH client must be configured to run as a master client, which is not common. Unfortunately, SSH master mode configuration does not require elevated privileges so an attacker can silently change the configuration and prepare a malicious binary that launches the original SSH client in the master mode, in a way similar to our design of ShadowMove on Linux (Section 3.4.3). We note that this lateral movement technique can overcome some limitations of ShadowMove because it can abuse SSH that employs payload encryption. Therefore, it is complementary to ShadowMove. Having said that, it is a specific technique that only works for SSH in a particular scenario, while ShadowMove is a general lateral movement technique.

ShadowMove can sniff traffic, but it is different from other traditional sniffing techniques: instead of eavesdropping on the network, ShadowMove sniffs traffic on the host; instead of capturing packets at the kernel level (like what WireShark does), ShadowMove sniffs traffic at the user level. Lateral movement usually involves privilege escalation or harvesting of additional credentials [22]. ShadowMove does not rely on either privilege escalation or credential harvesting, so it is a new type of lateral movement.

## 8 Conclusion

We propose the ShadowMove strategy that allows APT attackers to make stealthy lateral movements within an enterprise network. Built upon a novel socket duplication technique, ShadowMove leverages existing benign network connections and does not require any elevated privilege, new connections, extra authentication, or process injection. Therefore, it is capable of evading the detection of host- and network-level defensive mechanisms. To confirm the feasibility of our approach, we have developed a prototype of ShadowMove for modern versions of Windows and Linux OSes, which successfully abuses three common enterprise protocols (i.e., FTP, Microsoft SQL, and WinRM) for lateral movement, such as uploading malware to the next target machine and starting the malware execution on the next target. We describe the technical challenges in ShadowMove, such as how to generate network packets that fit in the context of an existing network connection. We also experimentally confirm that our prototype implementation is undetectable by state-of-the-art antivirus products, IDSes (such as Snort), and Endpoint Detection and Response systems. Our experience raises the bar for lateral movement detection in an enterprise environment and calls for innovative solutions.

## 9 Acknowledgement

We thank our shepherd Giancarlo Pellegrino and the anonymous USENIX Security reviewers for their insightful comments and suggestions that help improve the quality of this paper. This research has been partially funded by National Science Foundation by CISE's CNS-1566443 and gifts or grants from Fujitsu. Special thanks also go to Dr. Qiong Cheng at UNC Charlotte who shared her expertise in SQL scripting.

## References

[1] winrm for go library. https://github.com/masterzen/winrm. Accessed November 2018.

[2] winrm4j. https://github.com/cloudsoft/winrm4j. Accessed November 2018.

[3] Yama linux security module. https://www.kernel.org/doc/Documentation/security/Yama.txt. Accessed June 2019.

[4] Calling external program on application server. https://answers.sap.com/questions/7641883/calling-external-program-on-application-server.html, 2010. Accessed August 2019.

[5] List of ftp commands. https://en.wikipedia.org /wiki/List_of_FTP_commands, 2018. Accessed February 2019.

[6] Video Clip for the FTPShadowMove. http://54.36.162.222/ShadowMoveDemo/FTPShadowMove.gif, 2018.

[7] A Comprehensive Open Source Security Platform. https://wazuh.com/product/, 2019. Accessed October 2019.

[8] Install npm packages globally without sudo on macOS and Linux. https://github.com/sindresorhus/guides/blob/master/npm-global-without-sudo.md, 2019. Accessed October 2019.

[9] OpenSSH/Cookbook/Multiplexing. https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Multiplexing, 2019. Accessed October 2019.

[10] OSSEC: The World's Most Widely Used Host-based Intrusion Detection System. https://github.com/ossec/ossec-hids, 2019. Accessed October 2019.

[11] Service name and transport protocol port number registry. https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml, 2019.

[12] SQL powered operating system instrumentation, monitoring, and analytics. https://github.com/osquery/osquery, 2019. Accessed October 2019.

[13] ssh_config — OpenSSH SSH client configuration files. http://manpages.ubuntu.com/manpages/bionic/man5/ssh_config.5.html, 2019. Accessed October 2019.

[14] SWI Prolog. https://www.swi-prolog.org/, 2019. Accessed October 2019.

[15] Video Clip for the FTPShadowMove Demo on Ubuntu. http://54.36.162.222/ShadowMoveDemo/LinuxShadowMove.gif, 2019.

[16] Video Clip for the ShadowMove Demo. http://54.36.162.222/ShadowMoveDemo/ShadowmovePrototypeDemo.mp4, 2019.

[17] Video Clip for the SQLShadowMove Demo. http://54.36.162.222/ShadowMoveDemo/SQLShadowMove.gif, 2019.

[18] Adam Blaszczyk. Can we stop detecting mimikatz please? http://www.hexacorn.com/blog/2019/02/03/can-we-stop-detecting-mimikatz-please/, 2019. Accessed Feb 2019.

[19] Adam Boileau. Trust Transience: Post Intrusion SSH Hijacking. In *BlackHat Briefings*, August 2005.

[20] Thanh Bui, Siddharth Prakash Rao, Markku Antikainen, Viswanathan Manihatty Bojan, and Tuomas Aura. Man-in-the-machine: Exploiting ill-secured communication inside the computer. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1511–1525, Baltimore, MD, 2018. USENIX Association.

[21] Microsoft Windows Dev Center. Protecting Anti-Malware Services. https://docs.microsoft.com/en-us/windows/desktop/services/protecting-anti-malware-services-, 2018. Accessed August 2019.

[22] Ping Chen, Lieven Desmet, and Christophe Huygens. A study on advanced persistent threats. In Bart De Decker and André Zúquete, editors, *Communications and Multimedia Security*, pages 63–72, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[23] Weiteng Chen and Zhiyun Qian. Off-path TCP exploit: How wireless routers can jeopardize your secrets. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1581–1598, Baltimore, MD, 2018. USENIX Association.

[24] B. Deply. Mimikatz. https://github.com/gentilkiwi/mimikatz, 2014. Accessed February 2019.

[25] S. Duckwall and C. Campbell. Hello, my name is microsoft and i have a credential problem. In *Blackhat USA 2013 White Papers*, 2013. https://media.blackhat.com/us-13/US-13-Duckwall-Pass-the-Hash-WP.pdf.

[26] John Dunagan, Alice X. Zheng, and Daniel R. Simon. Heat-ray: Combating identity snowball attacks using machine learning, combinatorial optimization and attack graphs. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, pages 305–320, New York, NY, USA, 2009. ACM.

[27] James Forshaw. Injecting Code into Windows Protected Processes using COM - Part 1. https://googleprojectzero.blogspot.com/2018/10/injecting-code-into-windows-protected.html, October 2018. Accessed August 2019.

[28] Nalani Fraser, Jacqueline O'Leary, Vincent Cannon, and Fred Plan. Apt38: Details on new north korean regime-backed threat group. https://www.fireeye.com/blog/threat-research/2018/10/apt38-details-on-new-north-korean-regime-backed-threat-group.html, 2017.

[29] Steve Friedl. An Illustrated Guide to SSH Agent Forwarding. http://www.unixwiz.net/techtips/ssh-agent-forwarding.html, 2006. Accessed October 2019.

[30] gaffe23. Linux inject. https://github.com/gaffe23/linux-inject, 2016. Accessed July 2019.

[31] Tal Garfinkel and Mendel Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *Proceedings of Network and Distributed Systems Security Symposium (NDSS)*, February 2003.

[32] M. Haardt and M. Coleman. ptrace(2) Linux Programmer's Manual. http://man7.org/linux/man-pages/man2/ptrace.2.html, 1999. Accessed August 2019.

[33] Support Home. Clearing the Windows Temp Folders. http://lexisnexis.custhelp.com/app/answers/answer_view/a_id/1084415/. Accessed August 2019.

[34] CrowdStrike Inc. CrowdStrike Compromise Assessment Data Sheet. https://www.crowdstrike.com/wp-content/brochures/CrowdStrike_CompromiseAssessment_DataSheet.pdf, 2019. Accessed February 2019.

[35] A. D. Kent and L. M. Liebrock. Differentiating user authentication graphs. In *2013 IEEE Security and Privacy Workshops*, pages 72–75, May 2013.

[36] Linux. Linux ACL on shared memory objects. http://man7.org/linux/man-pages/man2/shmget.2.html. Accessed August 2019.

[37] Strategic Cyber LLC. Cobalt strike: Advanced threat tactics for penetration testers. https://cobaltstrike.com/downloads/csmanual38.pdf, 2017. Accessed February 2019.

[38] S. Metcalf. Unofficial guide to mimikatz & command reference. https://adsecurity.org/?page_id=1821, 2018. Accessed February 2019.

[39] Microsoft. Windows Remote Management. https://docs.microsoft.com/en-us/windows/desktop/WinRM/portal. Accessed November 2018.

[40] Microsoft. Duplicatehandle function. https://msdn.microsoft.com/en-us/library/windows/desktop/ms724251(v=vs.85).aspx, 2017. [Online; accessed 10-May-2018].

[41] Microsoft. Wsaduplicatesocket function. https://msdn.microsoft.com/en-us/library/windows/desktop/ms741565(v=vs.85).aspx, 2017. [Online; accessed 10-May-2018].

[42] Microsoft. Mib_tcprow2 structure. https://docs.microsoft.com/en-us/windows/desktop/api/tcpmib/ns-tcpmib-_mib_tcprow2, 2018. Accessed February 2019.

[43] Doug Miller, Ron Alford, Andy Applebaum, Henry Foster, Caleb Little, and Blake Strom. Automated adversary emulation: A case for planning and acting with unknowns. 2018.

[44] MSDN. [MS-TDS]: Tabular Data Stream Protocol. https://msdn.microsoft.com/en-us/library/dd304523.aspx, 2018. Accessed November 2018.

[45] MSDN. [MS-TDS]: SQL Batch Client Request. https://msdn.microsoft.com/en-us/library/dd304416.aspx, 2019. Accessed November 2018.

[46] Mark O'Neill, Scott Heidbrink, Jordan Whitehead, Tanner Perdue, Luke Dickinson, Torstein Collett, Nick Bonner, Kent Seamons, and Daniel Zappala. The secure socket API: TLS as an operating system service. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 799–816, Baltimore, MD, 2018. USENIX Association.

[47] Pradeep Padala. Playing with ptrace, part i. *Linux Journal*, 2002(103):5–, November 2002.

[48] Windows Defender Research. Detecting stealthier cross-process injection techniques with windows defender atp. https://cloudblogs.microsoft.com/microsoftsecure/2017/07/12/detecting-stealthier-cross-process-injection-techniques-with-windows-defender-atp-process-hollowing-and-atom-bombing/, 2019. Accessed Feb 2019.

[49] Ryan Ries. Monitoring with Windows Remote Management (WinRM) and Powershell Part I. https://www.myotherpcisacloud.com/post/Monitoring-with-Windows-Remote-Management-(WinRM)-and-Powershell-Part-I. Accessed November 2018.

[50] Neil J. Rubenking. The Best Antivirus Protection for 2019. https://www.pcmag.com/article2/0,2817,2372364,00.asp, 2019. [Online; accessed 04-February-2019].

[51] Hossein Siadati and Nasir Memon. Detecting structurally anomalous logins within enterprise networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1273–1284. ACM, 2017.

[52] Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. *Operating System Concepts*. Wiley Publishing, 9th edition, 2012.

[53] SolarWinds. SITE FTP command. https://support.solarwinds.com/SuccessCenter/s/article/SITE-FTP-command, 2017. Accessed August 2019.

[54] W. Richard Stevens, Bill Fenner, and Andrew M. Rudoff. *UNIX Network Programming, Vol. 1*. Pearson Education, 3 edition, 2003.

[55] Suricata. Suricata features. https://suricata-ids.org/features/, 2018. Accessed November 2018.

[56] Andrew S Tanenbaum and DJ Wetherall. Computer Networks, Fifth Edition. In *Pearson Education, Inc.* Prentice Hall, 2011.

[57] FireEye FLARE Team. Windows management instrumentation (wmi) offense, defense, and forensics. https://www.fireeye.com/content/dam/fireeye-www/global/en/current-threats/pdfs/wp-windows-management-instrumentation.pdf, 2015. Accessed February 2019.

[58] David Treadwell. socket.c. http://icerote.net/doc/library/programming/source/SOURCE.CODE.MICROSOFT.WINDOWS.2000.AND.NT4-BTDE/win2k/private/net/sockets/winsock2/wsp/msafd/socket.c, 1992. Accessed January 2019.

[59] David Treadwell. wspmisc.c. http://icerote.net/doc/library/programming/source/SOURCE.CODE.MICROSOFT.WINDOWS.2000.AND.NT4-BTDE/win2k/private/net/sockets/winsock2/wsp/msafd/wspmisc.c, 1992. Accessed January 2019.

[60] VMware. Configure WinRM to Use HTTP. https://docs.vmware.com/en/vRealize-Automation/7.5/com.vmware.vrealize.orchestrator-use-plugins.doc/GUID-D4ACA4EF-D018-448A-866A-DECDDA5CC3C1.html. Accessed November 2018.

[61] Taimour Wehbe, Vincent Mooney, and David Keezer. Hardware-Based Run-Time Code Integrity in Embedded Devices. *Cryptography*, 2(3), 2018.

## A Prepare the Environment for WinRM Hijacking

### A.1 Server Configuration

First, we configure the WinRM server on the remote machine by following these steps.

Set the default WinRM configuration

```
winrm quickconfig
```

Run the following command to check whether a listener is running, and verify the default ports

```
winrm e winrm/config/listener
```

Run the following command to enable basic authentication

```
winrm set winrm/config/service/auth
  '@{Basic="true"}'
```

Run the following command to allow transfer of unencrypted data by the WinRM server

```
winrm set winrm/config/service
  '@{AllowUnencrypted="true"}'
```

### A.2 Client Configuration

Next, we configure the WinRM client by following these steps.

Run the following command to enable basic authentication

```
winrm set winrm/config/client/auth
  '@{Basic="true"}'
```

Run the following command to allow transfer of unencrypted data by the WinRM client

```
winrm set winrm/config/client
  '@{AllowUnencrypted="true"}'
```

If the WinRM host machine is in an external domain, run the following command to specify the trusted hosts

```
winrm set winrm/config/client
  '@{TrustedHosts="host1, host2, host3"}'
```