Bro IDS Deployment (Bro-stallation) Guide

Contents

OVERVIEW

Bro is a packet monitoring and analysis framework that is highly programmable and extensible. In this respect, it is different than traditional IDS in that it provides both built-in functionality to detect suspicious network behavior and an development platform to incorporate custom analysis related to the specific business environment into which it is deployed.

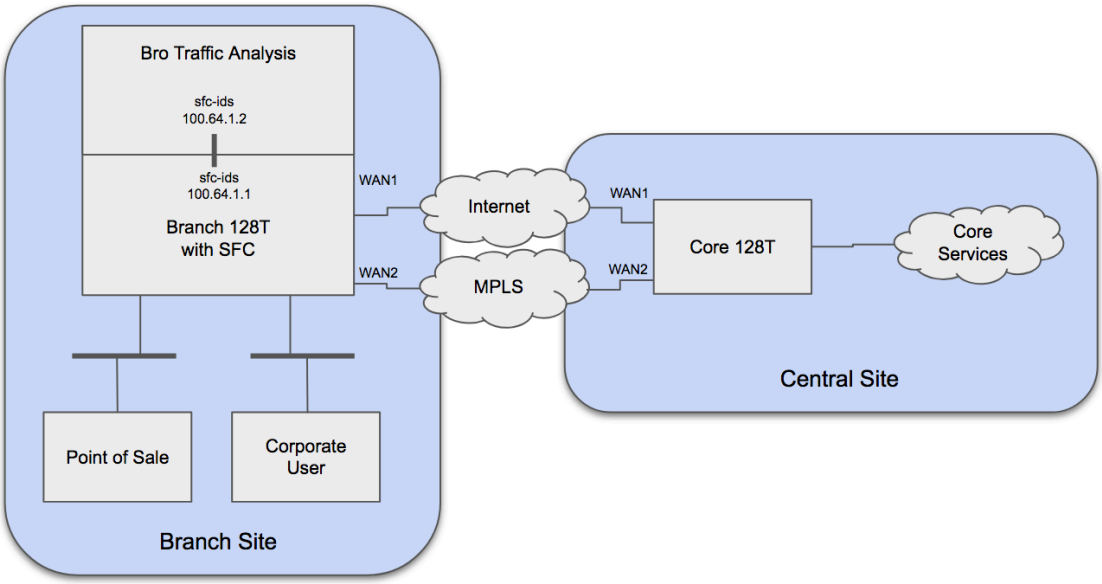
The 128 Technology solution is an open routing and security platform that enables network and security professionals to develop complete understanding and control of all network traffic. The 128T solution enables segmentation and steering of traffic using a business-focused data model that describes the tenants and services within the network.

Combining the 128T and Bro analysis solution, network and security teams are able to achieve a new and innovative level of visibility and control into the network and the business it supports.

DEPLOYMENT GOALS

Provide general guidelines around the deployment of an integrated, service chained Bro IDS into the 128T platform. Provide a development framework for analysis of network traffic including trigger and analytics data for observation and action.

PROTOTYPICAL NETWORK DESIGN



PLATFORM REQUIREMENTS

- Conductor for centralized management and visibility
- On premise or in public cloud
- Requires 2 core CPU, 8 GB RAM, 60 GB hard drive or greater
- One IP (public or private)
- Needs to be reachable by remote routers on the following ports:
- TCP port 930 (SSL connectivity from routers)
- TCP ports 4505-4506 (SaltStack connectivity from routers)

- Branch router(s)
- Appliance recommended for maximum performance
- Adequately sized Virtual Machine optional
- Minimum 2 ports for a standalone deployment
- Requires 4 core CPU, 8 GB RAM, 60 GB hard drive or greater
- One private IP for LAN data plane traffic
- One routable IP for WAN connectivity
    - NOTE: Software is installed directly using Internet-based repositories.

- Data Center router(s)
- Appliance recommended for maximum performance
- Requires 4 core CPU, 8 GB RAM, 60 GB hard drive or greater
- One private IP for LAN data plane traffic
- One public/routable IP for WAN connectivity

DEPLOYMENT MODEL

128T Routing software is open and extensible, providing the customer control over the host operating system while co-residing with 3rd party tools. The customer is ultimately responsible for the host operating system and package management while 128T co-resides additional applications.

There are two options for deployment of service chained software into the 128T routing node:

- Manual installation – For smaller or highly customized deployments, Bro may be installed manually. This document focuses on manual installation that may be automated as required.
- Automated installation – The 128T Conductor provides a set of DevOps tools to automate deployments in a repeatable fashion. Using SaltStack, the Bro IDS may be deployed and

managed by defining packages to be installed and files to be managed on the 128T routing nodes. Automated installation is currently out of scope of this document.

BRO INSTALLATION

Add the Bro repository and install Bro software:

```
cd /etc/yum.repos.d/
wget http://download.opensuse.org/repositories/network:bro/CentOS_7/network:bro.repo
dnf install bro
/opt/bro/bin/broctl deploy
```

Create a systemd service to ensure Bro is started upon creating of the service chain:

```
[root@localhost]# cat /etc/systemd/system/128t-sfc-setup.service

[Unit]
Description=Service to detect and setup Bro

[Service]
ExecStart=/usr/libexec/128t-sfc-setup.sh
Restart=always
RestartSec=5

[Install]

WantedBy=multi-user.target
```

Create the script to setup the Service Function Chain upon service execution:

```
[root@localhost]# cat /usr/libexec/128t-sfc-setup.sh

#!/bin/bash

#  Launch Bro
(
  cd /opt/bro/logs
  ip netns exec ids /opt/bro/share/broctl/scripts/run-bro -1 -i sfc-ids -U .status -p broctl -p broctl-live -p
standalone -p local -p bro local.bro broctl broctl/standalone broctl/auto
)


  echo "Launched IDS."
```

Create a plugin scripts to setup the Service Function Chain upon KNI initialization:

```
/etc/128technology/plugins/network-scripts/host/sfc-ids/init

#!/bin/bash

# Import common functions related to namespace operations
source /etc/128technology/plugins/network-scripts/common/kni_helpers
source /etc/128technology/plugins/network-scripts/common/namespace_helpers
source /etc/sysconfig/network-scripts/ifcfg-$1

INTF=$1
NS=$2

create_namespace $NS

# moving interfaces inside namespace
add_interface_to_namespace $NS $INTF
```

```
# admin up the relevant interfaces
namespace_interface_up $NS $INTF

#turn off reverse path filtering for host interfaces
namespace_execute $NS sysctl -w net.ipv4.conf.all.rp_filter=0 &>/dev/null
namespace_execute $NS sysctl -w net.ipv4.conf.$INTF.rp_filter=0 &>/dev/null
namespace_execute $NS sysctl -w net.ipv4.ip_forward=1

# set up KNI
namespace_execute $NS ip a add $IPADDR/$PREFIX dev $INTF
namespace_execute $NS ip route add default via 100.64.1.2 dev $INTF
```

/etc/128technology/plugins/network-scripts/host/sfc-ids/startup

```
#!/bin/bash

# Import common functions related to namespace operations
source /etc/128technology/plugins/network-scripts/common/namespace_helpers
source /etc/128technology/plugins/network-scripts/common/bash_helpers

INTF=$1
NS=$2


create_namespace $NS
```

/etc/128technology/plugins/network-scripts/host/sfc-ids/shutdown

```
#!/bin/bash

# Import common functions related to namespace operations
source /etc/128technology/plugins/network-scripts/common/namespace_helpers
source /etc/128technology/plugins/network-scripts/common/bash_helpers

INTF=$1
NS=$2

# move interfaces outside the namespace and delete namespace
remove_interface_from_namespace $NS $INTF
delete_namespace $NS 2>/dev/null
```

Log into Conductor and add KNI interfaces into IDS namespace (replace “Mobile128T” with router and node names:

```
config authority router Mobile128T node Mobile128T device-interface sfc-ids name               sfc-ids

config authority router Mobile128T node Mobile128T device-interface sfc-ids type               host

config authority router Mobile128T node Mobile128T device-interface sfc-ids network-namespace  ids

config authority router Mobile128T node Mobile128T device-interface sfc-ids network-interface sfc-ids name
sfc-ids

config authority router Mobile128T node Mobile128T device-interface sfc-ids network-interface sfc-ids global-id
31

config authority router Mobile128T node Mobile128T device-interface sfc-ids network-interface sfc-ids inter-router-
security  internal

config authority router Mobile128T node Mobile128T device-interface sfc-ids network-interface sfc-ids address
100.64.1.2 ip-address     100.64.1.2

config authority router Mobile128T node Mobile128T device-interface sfc-ids network-interface sfc-ids address
100.64.1.2 prefix-length  30
```

```
config authority router Mobile128T node Mobile128T device-interface sfc-ids network-interface sfc-ids address
100.64.1.2 gateway        100.64.1.1
```

Assign tenant to the IDS KNI neighborhood to classify traffic to forward. This will classify traffic returning from the IDS once inspected. There are two options to define IDS tenants:

1. Interface-based tenant on the IDS return KNI interface. All traffic will be treated equally with respect to tenancy once it has traversed the IDS:

```
config authority router Mobile128T node Mobile128T device-interface sfc-ids network-interface sfc-ids tenant
LAN

config authority tenant LAN name    LAN
```

2. Tenant-specific mask to ensure all traffic is inspected and classified into source tenants. These individual tenants may be assigned to services for segmentation, policy and routing, similar to scenarios with no IDS inserted:

```
config authority router Mobile128T node Mobile128T device-interface sfc-ids network-interface sfc-ids
neighborhood ids name      ids

config authority router Mobile128T node Mobile128T device-interface sfc-ids network-interface sfc-ids
neighborhood ids topology  spoke

config authority tenant PCI name    PCI
config authority tenant PCI member PCI neighborhood  ids
config authority tenant PCI member PCI address       10.0.50.0/24
config authority tenant GuestWiFi name    GuestWiFi
config authority tenant GuestWiFi member GuestWiFi neighborhood  ids
config authority tenant GuestWiFi member GuestWiFi address       192.168.101.2/32
```

Define a set of Services that are reachable once the traffic is inspected:

```
config authority service Server2 access-policy ids source  ids
config authority service Server3 access-policy ids source  ids
```

Define the tenant traffic that should be inspected. For branch deployments, this is typically the tenant LAN traffic :

```
config authority tenant LAN name    LAN
config authority tenant LAN member LAN neighborhood  LAN
config authority tenant LAN member LAN address       0.0.0.0/0


config authority service ids name                  ids
config authority service ids service-group         IDS
config authority service ids security              internal
config authority service ids address               0.0.0.0/0
config authority service ids access-policy LAN source  LAN
config authority service ids share-service-routes  false
```

Define service route to chain traffic on SFC KNI:

```
config authority router Mobile128T service-route ids name          ids
config authority router Mobile128T service-route ids service-name  ids
config authority router Mobile128T service-route ids next-hop Mobile128T sfc-ids node-name   Mobile128T
config authority router Mobile128T service-route ids next-hop Mobile128T sfc-ids interface   sfc-ids
config authority router Mobile128T service-route ids next-hop Mobile128T sfc-ids gateway-ip  100.64.1.1
```

Now that the KNI interfaces are in place, Restart 128T, ensure systemd script is executable and enable the service:

```
sudo chmod +x /usr/libexec/128t-sfc-setup.sh
sudo systemctl enable 128t-sfc-setup
sudo systemctl start 128t-sfc-setup
```

Reboot 128T to confirm all services restart as expected.

```
sudo reboot
```

SAMPLE LOGS

The following are some sample logs from the /opt/bro/logs directory:

```
-rw-r--r-- 1 root bro    254 Mar 10 18:28 capture_loss.log
-rw-r--r-- 1 root bro  47629 Mar 10 18:33 conn.log
lrwxrwxrwx 1 root bro     18 Mar 10 18:10 current -> /opt/bro/spool/bro
-rw-r--r-- 1 root bro  76432 Mar 10 18:36 dns.log
-rw-r--r-- 1 root bro 105348 Mar 10 18:28 files.log
-rw-r--r-- 1 root bro  15952 Mar 10 18:27 http.log
-rw-r--r-- 1 root bro    177 Mar 10 18:14 known_hosts.log
-rw-r--r-- 1 root bro  24367 Mar 10 18:13 loaded_scripts.log
-rw-r--r-- 1 root bro    898 Mar 10 18:27 notice.log
-rw-r--r-- 1 root bro    226 Mar 10 18:13 packet_filter.log
-rw-r--r-- 1 root bro    546 Mar 10 18:27 software.log
-rw-r--r-- 1 root bro  51643 Mar 10 18:28 ssl.log
-rw-r--r-- 1 root bro   1105 Mar 10 18:33 stats.log
-rw-r--r-- 1 root bro   4360 Mar 10 18:15 weird.log
-rw-r--r-- 1 root bro  55700 Mar 10 18:28 x509.log
```

TUNING BRO

Note that this document describes adding a single instance, single processor of Bro to a branch device. There are several modifications that may be made to the Bro package installation that may be of benefit. The following are outside of the scope of the current document, but may be added in a future version:

1. Pinning of Bro to a particular CPU that is not handing 128T forwarding.
2. Multi-threading Bro with multiple instances.
3. Compiling Bro with GeoIP support added at run time.

SAMPLE BRO-CUTS

Once logs have been accumulated, bro-cut provides a quick method to analyze and extract session data.

The bro-cut tool allows a list of log headers to be specified and searched using tools such as awk. Some examples of queries to find http sessions and Microsoft-related session information is shown in the queries below.

Set path and log directory:

```
PATH=$PATH:/opt/bro/bin
cd /opt/bro/logs
```

Show HTTP sessions and destinations:

```
bro-cut service id.resp_p id.resp_h < /opt/bro/logs/conn.log | awk '$1=="http" && !($2==80 || $2==8080)
{ print $3 }' | sort –u
```

Show all invalid certificates and their destination hosts:

```
bro-cut id.resp_h id.resp_p note < /opt/bro/logs/notice.log | awk '$3~"Invalid_Server_Cert" { print
"address " $1 " port " $2 }'
```

Find all port scans executed against routed hosts:

```
find . -name notice.*.log.gz -exec zcat "{}" + | /opt/bro/bin/bro-cut -d ts note src | awk '$2~"Scan" {
print $1 " " $2 " address: " $3 " Port Scan!" }' | sort –u
```

Find unencrypted Microsoft update traffic:

```
find . -name http.*.log.gz -exec zcat "{}" + |  /opt/bro/bin/bro-cut -d ts id.resp_h host uri
response_body_len | awk '$3~"update" || $3~"microsoft" { print $1 " " $2 " dest: " $3 " URL: " $4 "
length " $5 }' | sort -u
```

DISCLAIMER

The information in this document is non-binding and for informational purposes only. It is subject to adjustment or modification after review, consideration, and negotiation of the terms and conditions of a definitive contract. All data and information contained herein is to be considered confidential and proprietary. The data and information contained herein may not be reproduced, published, or distributed to, or for, any third parties without the express prior written consent of 128 Technology.

www.128technology.com | info@128technology.com