OFTG-Ninja User Guide

Find a way out

Version 0.4

Ryan O'Horo

IOActive, Inc. 2015

CONTENTS

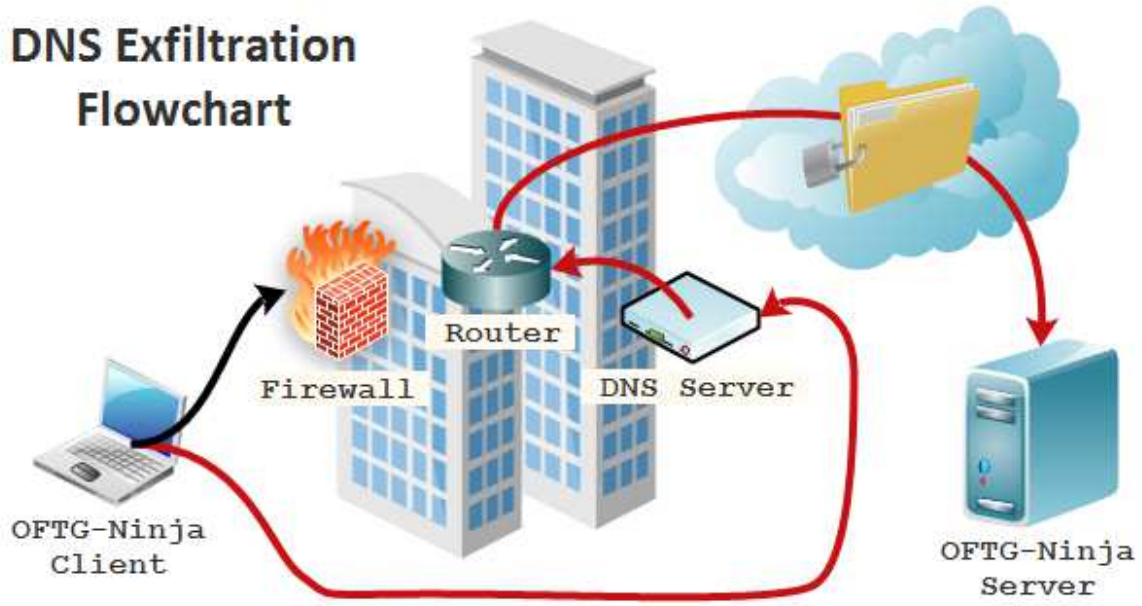
Overview

About OFTG-Ninja

OFTG-Ninja's goal is to be a vulnerability assessment tool for DLP appliances, firewalls and policies that prevent certain types of data from leaving a network. By utilizing a suite of exfiltration methods, a large number of possible exfiltration vectors are tested and actively exploited.

How OFTG-Ninja Works

OFTG-Ninja uses a client-server model to move data from one network to another. Using packet capture and forgery, packet transmission over various protocols are simulated without the need for full protocol stacks or bound ports. This frees host ports for other services and allows any protocol to interact with OFTG-Ninja.



Technical Requirements

OFTG-Ninja is a python application which uses an embedded web server to allow user interaction. Because OFTG-Ninja is a full web server and packet processing application, several dependencies need to be met before running.

OFTG-Ninja will consume CPU resources proportional to the amount of network traffic received. To get optimal performance, it's recommended that high-bandwidth applications be paused while OFTG-Ninja is running tasks.

PREREQUISITES

The following components are required for and have been fully tested with OFTG-Ninja:

- Operating Systems
  - Windows 7, Server 2012
- Application Components
  - Python 2.7.8, 2.7.9
  - WinPcap 4.1.3
- Python Modules
  - Python Flask 0.10.1
  - Python Flask-SocketIO 0.6.0
  - Python Websocket-Client 0.32
  - Python socketIO-client 0.6.5
  - Python Impacket 0.9.13
  - Python PyCrypto
  - Python Dropbox 3.36
  - Python TwitterAPI 2.3.3.4
  - Python Hexdump 3.2

INSTALLATION

Extract the OFTG-Ninja archive to its working directory.

NOTE: SSL support is not available in version 0.4 due to limitations of some dependencies.

NOTE: The SSL certificate that is included with OFTG-Ninja is self-signed. The certificate must be replaced with a valid one before using OFTG-Ninja remotely. Save server.key and server.crt to the OFTG-Ninja root directory.

GETTING STARTED

STARTING OFTG-NINJA

To start OFTG-Ninja, from the installation directory, execute the following command.

```
python oftg-ninja.py
```

NOTE: OFTG-Ninja must be run as an Administrator or root user to enable raw socket use and promiscuous packet capture.

OFTG-Ninja will prompt you to enter some basic configuration information if this is the first time you've started the application. Create a user to access the web application.

```
* Starting OFTG-Ninja v0.4 ...
To access OFTG-Ninja, you must first create a user...
Username: admin
```

```
Adding new user admin
Password:
```

To access the OFTG-Ninja web interface, navigate to the following URL.

```
http://127.0.0.1:7331/
```

Deploying a Server

In order to capture data sent by the Client, a Server instance must be started outside the network under test.

The Server host should be configured as such that no firewall, IPS or other appliance will interfere with incoming traffic, and should be on a network outside of the network being assessed. While OFTG-Ninja does most of its work passively, the application may interfere with services already running on the server host by sending specially crafted network packets. Do not deploy OFTG-Ninja on a host running production services to minimized the potential for issues.

Configuring Prerequisite Resources

Some plugins require ancillary services or configuration values that will allow the plugin to function as intended. See Cases.

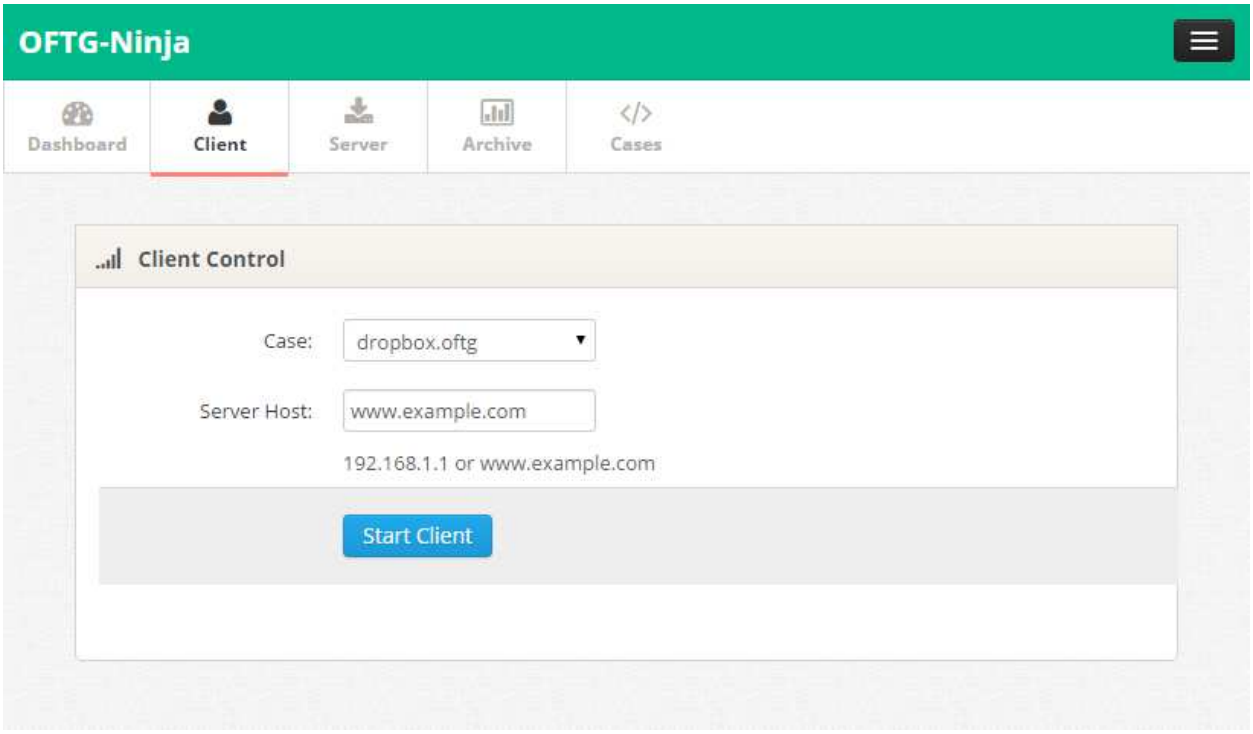
Examples:

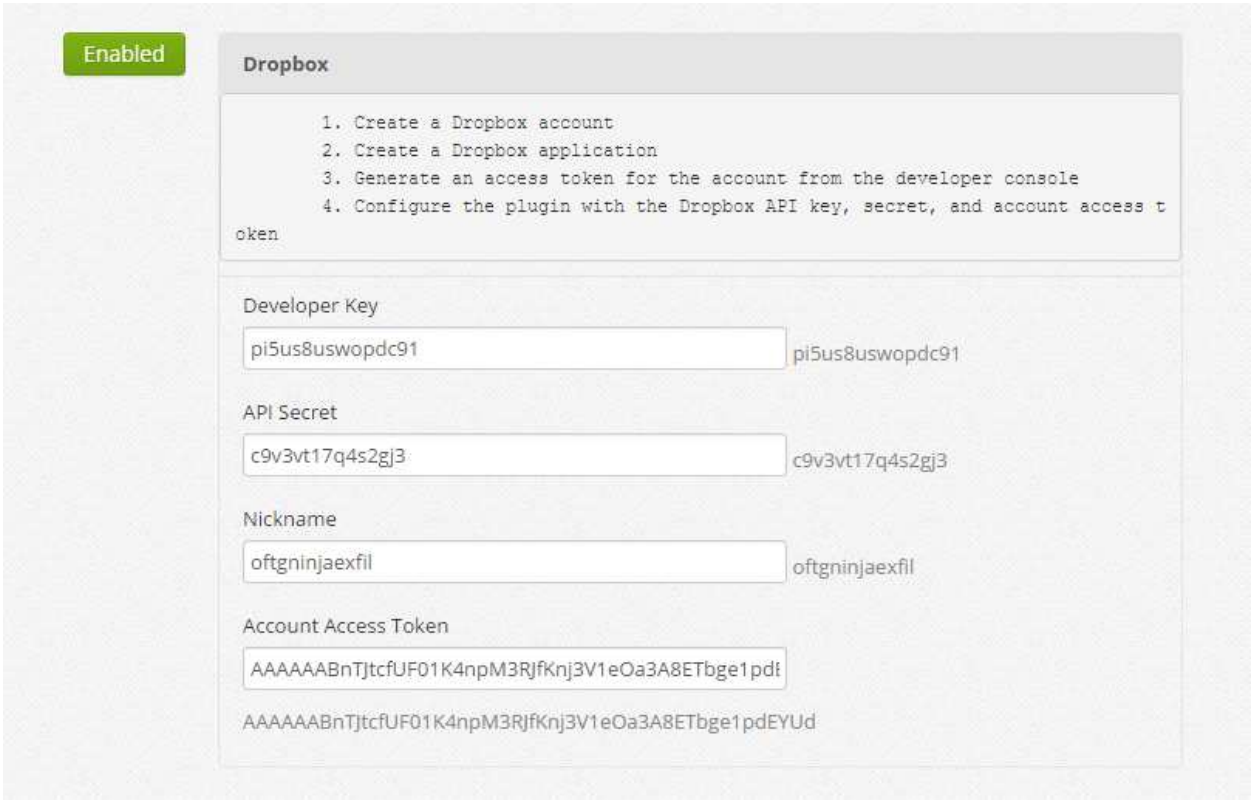
The DNS Plugin requires a valid domain under which a name server exists which can receive DNS requests from the Client. A dedicated domain name should be configured with the address of the Server host as the name server for that domain.

Plugins which use 3rd-party APIs may require developer keys. These keys need to be generated by the API service and configured in OFTG-Ninja before they will function properly.

Configuring a Client

After installing and running OTFG-Ninja on the Client host, open the web interface and create a new Case in the Cases tab. Select the Plugins you wish to use using the Enable buttons for this Case and customize the Plugin parameters for your environment. Upload the payloads intended for testing, if desired. Configure any other desired Case options like Compression and Encryption. Click Save Case.


OFTG-Ninja
Dashboard
Client
Server
Archive
Cases
Client Control
Case:
dropbox.oftg
Server Host:
www.example.com
192.168.1.1 or www.example.com
Start Client


Enabled
Dropbox
1. Create a Dropbox account
2. Create a Dropbox application
3. Generate an access token for the account from the developer console
4. Configure the plugin with the Dropbox API key, secret, and account access token
Developer Key
pi5us8uswopdc91
pi5us8uswopdc91
API Secret
c9v3vt17q4s2gj3
c9v3vt17q4s2gj3
Nickname
oftgninjaexfil
oftgninjaexfil
Account Access Token
AAAAAABnTJtcfUF01K4npM3RJfKnj3V1eOa3A8ETbge1pdE
AAAAAABnTJtcfUF01K4npM3RJfKnj3V1eOa3A8ETbge1pdEYUd

RUNNING AN ASSESSMENT

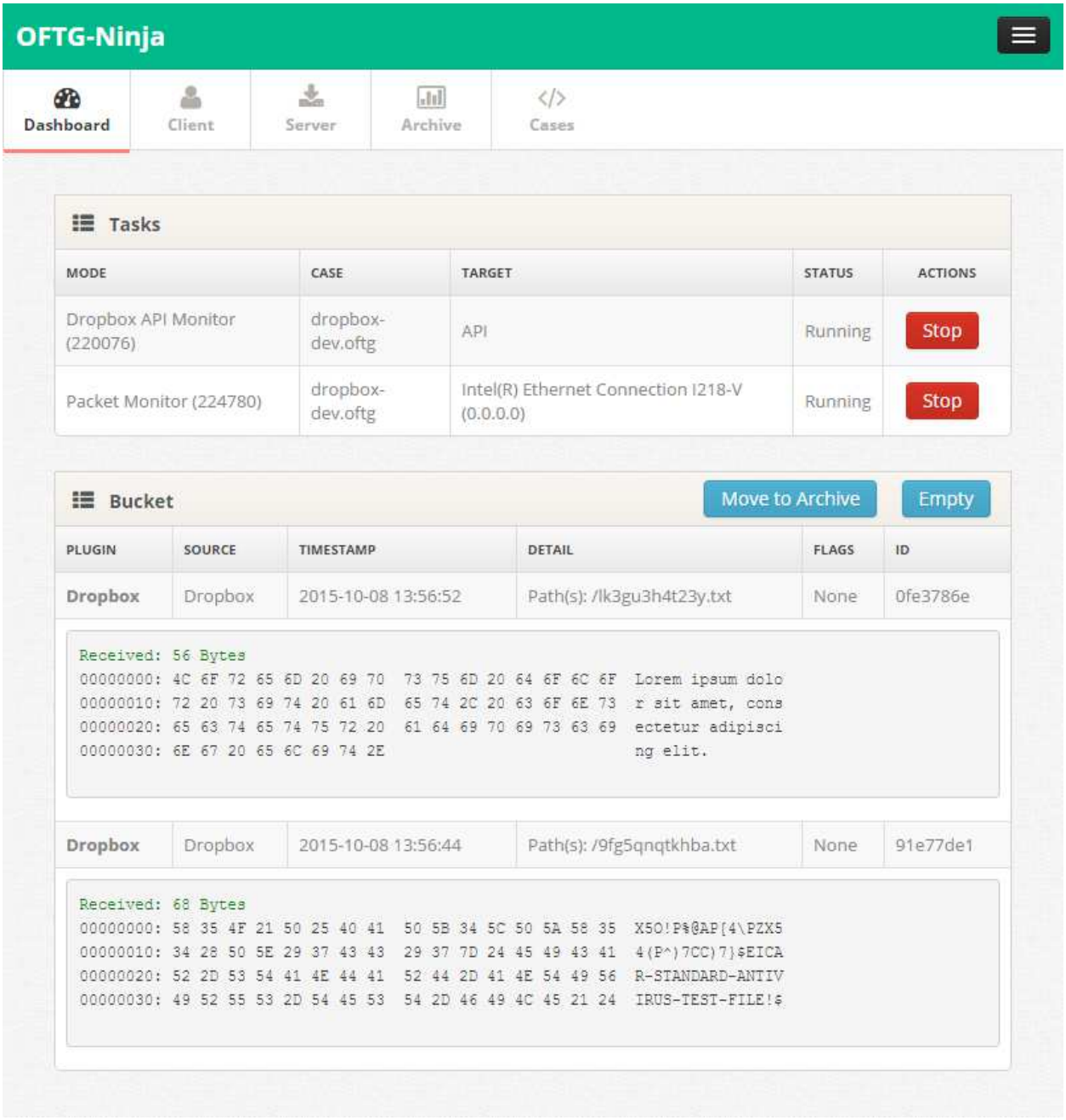
You'll want to copy the Case file created by the client to the Server so that the configurations match. Each Case configuration parameter is used to identify traffic generated by the Client, and so they should be identical.

Open the web interface on the Server host and navigate to the Server tab. Choose the interface in the drop-down menu from which you intend to receive Client traffic on. Choose the Case you want to use for this Server instance.

Click Start Server. A new Server task will be started.

This will start the packet capture process and load all available plugins for IP traffic analysis, as well as any configured API monitoring plugins.

With the Server Task running on the Server host, open the web interface of the Client host and navigate to the Client tab. Choose a Case and enter the external IP address(s) of the Server host. Click Start Client. One or more Client Tasks will be created. While these Tasks are running, the Client will iterate through each selected Plugin attempting to contact the Server.



As the Client runs, The Server will collect any available traffic generated by the Client. The Server will display data received in the Bucket and details about which Plugin was used to send it.

CASES

PAYLOADS

Each case can have an arbitrary number of payloads associated with it. Each payload is sent by the client once for each plugin iteration. For example, if you selected the UDP Plugin with 3 tested ports and 2 payloads, a total of 6 payload bodies would be sent. These payload bodies would then be split if their size exceeded the maximum data size for a single packet.

Payloads are loaded into OFTG-Ninja in the form of individual files and are stored inside the Case file.

NOTE: It is recommended that payload files not exceed 500MB.

PLUGINS

OFTG-Ninja offers a plugin framework which allows the application to utilize various prescribed methods of moving data between the client and the server.

BUILT-IN PLUGINS

UDP – Adds payloads to the data section of UDP packets.

ICMP – Adds payloads to the data section of ICMP packets.

DNS – Adds payloads to the hostname portion of DNS queries.

Dropbox – Uses HTTPS to upload payloads to a Dropbox account.

Twitter/Pastebin – Uses a combination of HTTP and HTTPS to upload a payload to Pastebin, then link to the payload via Twitter.

Users can create their own plugins by creating a class file which inherits the `OFTGPluginPacket` or `OFTGPluginAPI` classes. User classes should implement the collector and emitter methods, as well as utilize the available encoder method to properly tag and chunk payload data before sending or after receiving data. User plugins should be stored in the `plugins` subdirectory.

```
import socket
from classes.oftgplugin import OFTGPacketPlugin

class OFTGSample(OFTGPacketPlugin):
```

The `emitter` method should utilize the `encoder` method which returns an iterable of payload chunks which are tagged with metadata and appropriately sized for the medium.

```
def emitter(self)
     for chunk in self.encoder(self.payload):
          ...
          socket.sendto(chunk, (self.target, port))
```

The `collector` method takes a string which contains the data of a single unfiltered packet. The `collector` method must filter and dissect the packet data to assure the packet is intended to be processed by this plugin. The `decoder` method will parse the payload body, verify the checksum in the metadata and return a `dict` of relevant information if the payload body is valid OFTG-Ninja data. The `collector` method must return the `dict` provided by `decoder`, however the user may add or change parameters before doing so.

```
def collector(self, packetbuffer):
     payload = self.decoder(mydissector(packetbuffer))
     return payload
```

Plugins which subclass `OFTGAPIPlugin` must define a method `apifilter`. This method is called by OFTG-Ninja when the specified API returns some data for processing and returns the decoded payload. The `collector` method must also return a `list` consisting of: an `iterator` which returns API data, a `string` label and the user-defined `apifilter` method.

```
def collector(self, packetbuffer):

     def apifilter(apiresponse):
          payload = base64.b64decode(res.text)
          result = self.decoder(payload)
          return result

     ret = ExampleAPI(self.PROPERTIES['api_secret'])
     return [ret.get_iterator(), 'APIEntry', apifilter]
```

The plugin should define a `dict INFO` and `dict PROPERTIES` so OFTG-Ninja knows how the plugin functions and what parameters are required to configure it. Valid values for `Type are Boolean`, `string`, `files` and `list`. List type properties must also include a comma-delimited entry `List`. `INFO` must include `Title`. `PROPERTIES` must include entries `Label`, `Default`, `Sample`, `Type`, and `Value`. Entry `Label` must not be `None`.

```
INFO = {
    'Title': 'Example Plugin',
    'Usage': 'User guide plugin sample',
    'Author': 'Example User'
}

PROPERTIES = {'portspec':
                  {'Label': 'Ports',
                   'Default': '1-65535',
                   'Sample': '23,25,137-139',
                   'Type': 'string',
                   'Value': None},
              'portrandomize':
                  {'Label': 'Randomize Ports',
                   'Default': True,
```

```
                    'Type': 'boolean',
                    'Value': None}
                'protocol':
                    {'Label': 'Protocol',
                    'Default': 'TCP',
                    'Type': 'list',
                    'List': 'TCP,UDP',
                    'Value': None}
    }
```

The plugin should also define `dict DATASIZE`. This value is the maximum practical payload size for the given protocol. It's used to limit the chunk size of large payloads and is used by the `encoder` method.

```
DATASIZE = 65507
```

CONFIGURATION FILE

The configuration file, which can be specified with the –c argument, contains the basic startup parameters for OFTG-Ninja.

```
[webserver]
http_port = 7331
host_addr = 127.0.0.1

[users]
admin =
c56d8fa9593e7b6f66a97dd3d9a64060fb819797b848a55e2835dfbd4f570bd8
user =
3af40bd946ed271d6d539b3e415206b0772938a997d03e1e46a0e81a6d62410f
```

Change the host_addr parameter to 0.0.0.0 to force OFTG-Ninja to bind to an external adapter for remote access. Assure OFTG-Ninja has been configured with a secure password before enabling remote access to the web interface.

New users should be added via the command line interface and not modified directly in the configuration file.