ARBOR®
NETWORKS

The Security Division of NETSCOUT

PyNetSim
A modern INetSim Replacement

Jason Jones

FIRST 2017

BackGround

Why?

- Research teams may need a simulated environment because
  - They are not allowed to directly contact malware C2s
  - Trying to avoid tipping off threat actors
  - Command-and-control servers are down
- DNS redirection isn't enough
  - Hard-coded DNS servers still circumvent
  - Hard-coded IP addresses in lieu of DNS
- Internet simulation also allows for
  - Possibility of collecting client communications used to develop signatures
  - Keeping malware alive in memory long enough to take memory snapshots for static analysis
  - Test protocol re-implementation for a botnet monitoring system
  - QA of parsing / implementation of intelligence feeds
  - Possibility to direct actions of executed malware to activate certain pieces of code

Existing Solutions

INetSim

- The Original Internet Simulator
- Written in Perl ☹
- Built-in traffic redirection support relies on obsolete ip_queue support in Linux kernel
  - Other ways to get around this
- Significant Protocol Support
  - HTTP(S)
  - SMTP(S)
  - POP3(S)
  - FTP(S)
  - DNS
  - TFTP
  - IRC
  - Others

FakeNet-NG

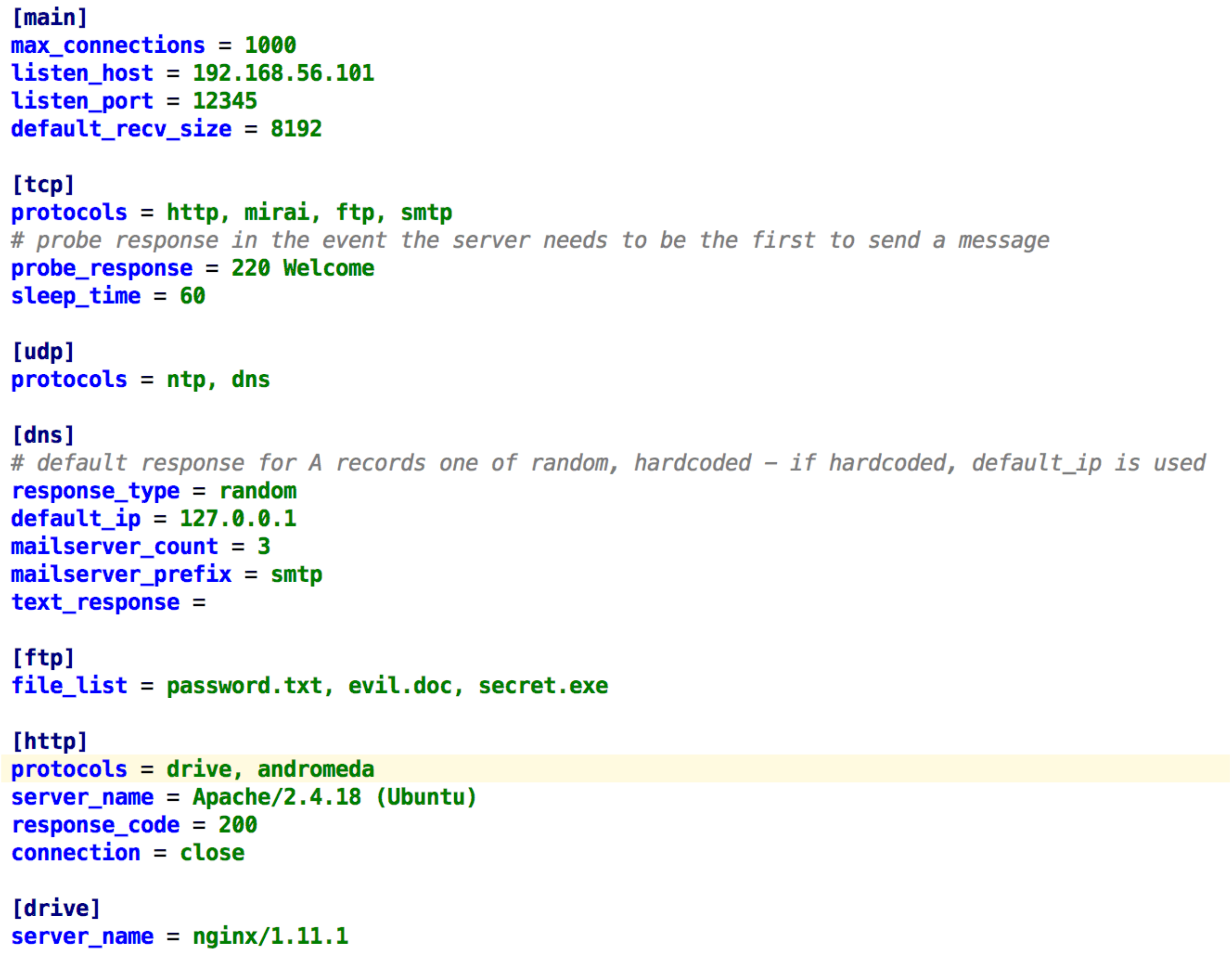
- Released by FireEye's FLARE team at BlackHat USA 2016
  - https://github.com/fireeye/flare-fakenet-ng/
  - Actively maintained
- Supports multiple protocols + SSL on most protocols
  - TFTP
  - SMTP
  - POP
  - IRC
  - HTTP
  - FTP
  - DNS
- Windows + Linux support (only recently learned of Linux support)
  - AFAICT, no dynamic protocol / SSL support
  - Doesn't speak malware protocols

PyNetSim

PyNetSim

- Built using Python3
- Goal is to dynamically detect the TCP / UDP protocol used, examples:
  - Detect HTTP on non-standard ports
  - Detect FTP on non-standard ports
  - Detect TLS/SSL enabled connections on non-standard ports
- Attempt to detect malware protocol used and speak that protocol
  - Allows for “proper” responses to keep an infected system “talking”
    - Keeps malware running for memory forensics, debugging purposes
    - Directed execution via commands sent back
  - Example: Alina requires a non-standard HTTP status code of 666 in the response
  - Example: Mirai CnCs have increasingly used ports 80 and 443 to evade port-based blocking, but is very recognizable in comparison to HTTP / HTTPS

Configuration



```
[main]
max_connections = 1000
listen_host = 192.168.56.101
listen_port = 12345
default_recv_size = 8192

[tcp]
protocols = http, mirai, ftp, smtp
# probe response in the event the server needs to be the first to send a message
probe_response = 220 Welcome
sleep_time = 60

[udp]
protocols = ntp, dns

[dns]
# default response for A records one of random, hardcoded - if hardcoded, default_ip is used
response_type = random
default_ip = 127.0.0.1
mailserver_count = 3
mailserver_prefix = smtp
text_response =

[ftp]
file_list = password.txt, evil.doc, secret.exe

[http]
protocols = drive, andromeda
server_name = Apache/2.4.18 (Ubuntu)
response_code = 200
connection = close

[drive]
server_name = nginx/1.11.1
```

Traffic Redirection

- Two ways to handle traffic redirection…
  - The hard way
    - Using NFQUEUE
    - NFQUEUE allows for incoming packets to be assigned to a queue that a listening program can consume from
    - Consuming the packets allows for parsing and manual response of things that would otherwise be rejected
    - Allows for keeping records of original address and port
    - Requires manual everything - handshakes, seq/ack calculation, ACKs, etc.
  - The easy way
    - Use built-in IPTABLES functionality
    - -j REDIRECT to send all ports from specified protocols to a single port
    - Now only need to listen on one port and let kernel take care of the rest
    - ~~Downside: lose the original address and port which may help to hint the protocol~~
    - Thanks to **Lauri from Cyber Defense Institute** for alerting me to SO_ORIGINAL_DST to allow for recovery of original destination and port

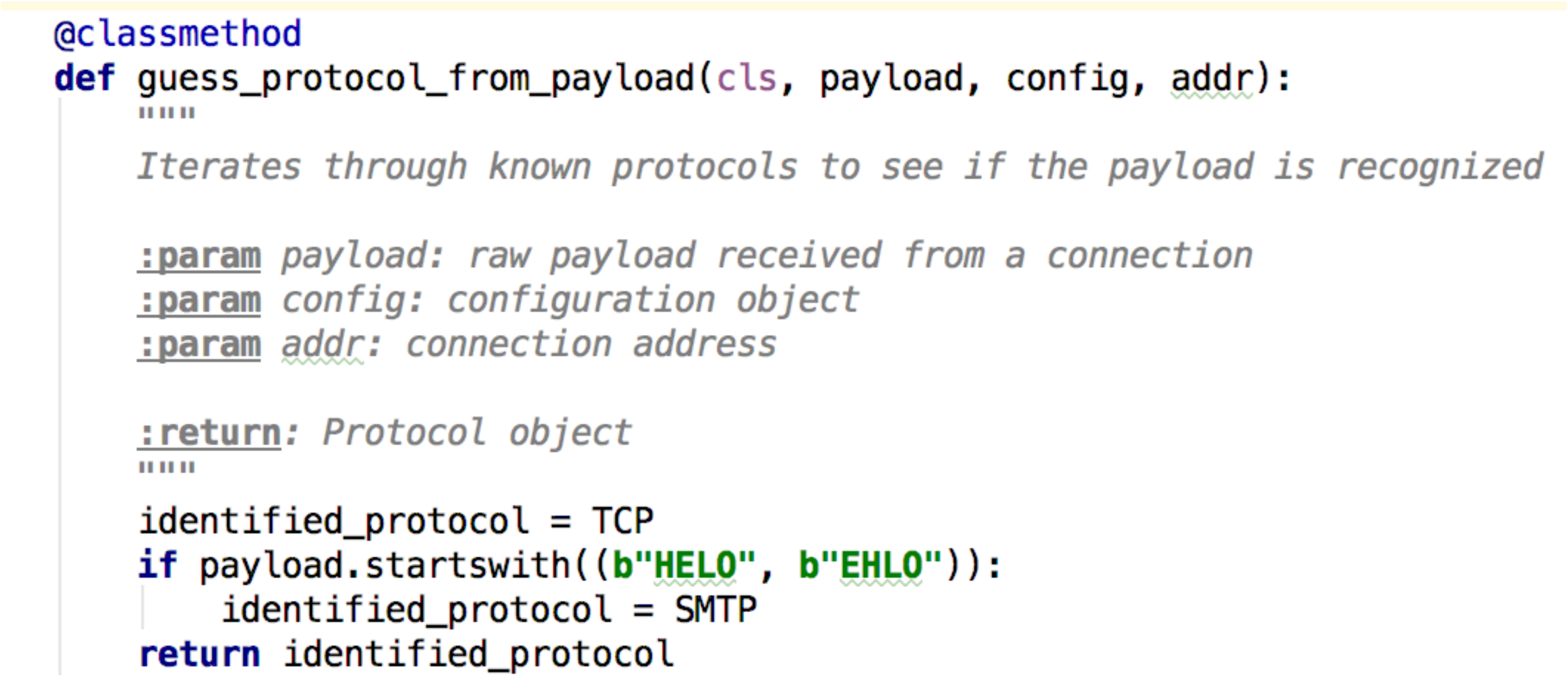
PyNetSim Protocols

- Targeting protocols that may be used by malware to communicate or exfiltrate data
- Dynamic SSL detection
  - Detect TLS Client Hello, use ssl.wrap_socket
- DNS – UDP & TCP
  - Respond with hardcoded or random non-RFC 1918 address
  - Responds to A, AAAA, MX types (bug in DPKT currently prevents TXT, bug filed)
- HTTP
  - DirtJumper / Drive families – ability to generate DDoS traffic
  - Andromeda – simple no-op back, TBD: add support for sending download commands for executables or plugins
- Telnet
  - In Progress
- SMTP
  - Supports STARTTLS, AUTH LOGIN + PLAIN, email sending support
- FTP
  - Simple login / listing / other commands, TBD => PASV / PORT support for commands like LIST / STOR / RETR
- IRC
  - In Progress
- Binary malware protocols
  - Mirai - simple hard-coded pings, optional ability to hard-code attack orders
  - LizardStresser - simple hard-coded pings, optional ability to hard-code attack orders

Dynamic Protocol Detection

- Inspired by scapy's "guess_payload" functionality for dissecting packets properly
- Peek at first payload, pass to known L7 protocol layers
  - Each high-level protocol may then opt to pass to child protocols
- First test for a TLS Client Hello
  - If detected, use ssl.wrap_socket and then continue checking the payload
- Each protocol has a set of defined child protocols in the config
- Each protocol has its own set of options to use
- Use dpkt where possible to help guess protocol using the parsing layers

Protocol Detection Example - SMTP



```
@classmethod
def guess_protocol_from_payload(cls, payload, config, addr):
    """
    Iterates through known protocols to see if the payload is recognized

    :param payload: raw payload received from a connection
    :param config: configuration object
    :param addr: connection address

    :return: Protocol object
    """
    identified_protocol = TCP
    if payload.startswith((b"HELO", b"EHLO")):
        identified_protocol = SMTP
    return identified_protocol
```

Demos

ARBOR NETWORKS

The Security Division of NETSCOUT

Conclusion

- Available on GitHub (soon)
  - https://github.com/arbor-jjones/pynetsim
- Future Work
  - Automated deployment via Dockerfile / Vagrant / ??
  - Solidify TLS / SSL support via dynamic generation of self-signed certs based on name in SNI
  - Pcap / payload export
    - Include decrypted SSL payloads
  - Web Inteface / Control Panel functionality
    - REST-ful API to query data
    - Allow for dynamic pushing of commands for supported protocols
    - Storage of keys for passing back to analyst / processing system to decrypt traffic
  - Better traffic redirection to use original port to hint
    - Also needed for handling passive FTP
  - Properly handle “special” DNS queries
    - SORBS / DNS-based blacklist checks used by malware like Sarvdap
    - Proper DNS exfiltration responses where required

ARBOR NETWORKS

The Security Division of NETSCOUT