**Remotely Extensible, Gmail-Interlinked, Decentralized Botnets**

Justin Dierking

**Abstract**

*I’m demonstrating a decentralized botnet that uses publicly accessible web services to connect and link together bots operating behind contemporary NATs and firewalls. Every bot is linked together using modular send and receive connectors that allow them to interact without the use of a centralized command-and-control system. In this demonstration, SMTP send and POP receive connectors have been integrated into each bot allowing them to communicate with each other using Google Gmail accounts. The goal of this project is to develop a message-switched dark-network that performs message forwarding and message routing with the use of an extensible, decentralized botnet.*

*Categories and Subjects*: Layout of Basic Architecture – Receive Connectors – Message Interpreting and Execution – Transmit Connectors

**Introduction**

Botnets are a collection of programs distributed across multiple computing systems that are networked together to accomplish a distributed task. The word, botnet, is derived from the terms robot and network. Traditionally, botnets are networked together into a hierarchy via IRC, HTTP, social networking, etc. A botnet is controlled from a central point, command-and-control (C&C). The C&C issues commands and modifications to each bot in a botnet.

Botnets are often employed for illicit activities. Such activities include email spamming, denial of service attacks, identity theft, unauthorized bitcoin mining, extortion, scareware, and malware deployment. The economic benefits to a developer/herder of a botnet can be substantial if they choose to sell access to their bots for implementing the before-mentioned attacks.

Today’s botnets are dismantled by identifying and taking down botnets’ C&Cs leaving the bots orphaned from control. This approach involves honey-potting a bot by purposefully allowing a computer to be infected and recruited into a botnet. By observing changes to the computer’s state after it has been recruited, the software that comprises the bot can be located and analyzed. Once the bot has been isolated on the system, recording and analyzing its behavior will yield details on its purpose and mode of communication with its C&C.



**Figure 1:** *Gmail interlinked botnet*

This project seeks to improve upon the general design of botnets by eliminating the use of C&Cs and hiding bot-to-bot messaging within indiscriminate services commonly used by the public today. The bots developed in this project use TLS enabled POP and SMTP to forward messages to other bots with the use of a Gmail accounts. A decentralized, message-switched, dark network is realized when multiple bots are networked together using multiple Gmail accounts. When arranged into a mesh topology, a redundant, fault-tolerant dark-network is formed.

**Layout and Basic Architecture**

The architecture of the bot’s design is organized to be modular and extensible. The operation of a bot is divided into three categories: receive connectors, message interpreting and execution, and transmit connectors. To process a message, a bot receives and parses a message with a receive connector. The parsed message is then read and interpreted by an interpreter. Depending on the contents of the message, the interpreter can forward a message or perform some other operation in its command set. If the interpreter forwards a message, it invokes a transmit connector to send a message to another bot.



**Figure 2:** *Process of receiving, processing, and sending messages.*

To construct a usable network, a bot will have multiple send and receive connectors. These connectors link a bot to other bots using the same Gmail accounts. Throughout a network, bots reference each other using aliases. If an alias can be reached on the same Gmail account, it is considered to be a local alias. If an alias can only be reached by forwarding a message through another local alias the alias is considered a remote alias.



**Figure 3:** *Bot using multiple connectors and Gmail accounts.*

The interpreter of a bot possesses a minimal command set. The command set includes functions for forwarding messages to local and remote aliases, copying files from one alias to another, and executing commands. To extend the bot’s functionality, files are written remotely to the system a bot resides on.

**Receive Connectors**

Receive connectors are programs that interact with a web service, download, and parse messages. Multiple connectors for various protocols and services can be used and run simultaneously. The only requirement for all receive connectors is that they generate output in a common format that the bot’s interpreter can read.

In the case of this project, a single receive connector has be developed for connecting to Gmail’s TLS enabled POP service. When this receive connector is executed, a session is established via TLS. Within the TLS session, the connector interacts with Gmail using the Post Office Protocol (POP) version 3 as specified in RFC 1939.

With a session open to Gmail’s POP, the authentication state of the session must be elevated. The “USER” and “PASS” commands are issued with a Gmail account’s username and password respectively. With the authentication state elevated, the receive connector may now access email for that Gmail account’s mailbox. Once the receive connector has access to the mailbox, it will use the LIST, TOP, and RETR commands to filter, retrieve, and delete email.



**Figure 4:** *Logging into Gmail via POP. Session captured from OpenSSL.*

First, the receive connecter executes a LIST to identify how many emails are in the mailbox and what index and size each email has. With the indexes known, the connector loops through each email checking the subject line of each email’s header. If the subject line matches what the connector is configured to filter for, the connector will download the raw email by executing the RETR command to retrieve and delete email from the mailbox.



**Figure 5:** *Logging into Gmail, listing emails, and displaying email via POP session captured from OpenSSL.*

Once an email has been downloaded, it is parsed. The process begins with scanning the header for content-type and content-transfer-encoding lines. The content type-line specifies whether an email is a single or multipart email. It also specifies the type of data in the body and the character-set the body is written in. The content-transfer-encoding line indicates what type of transform the original email has undergone to convert it to 7 bit ASCII. With the values of these two fields known, the body of the email can be parsed.

The current implementation of the Gmail POP receive connector can only parse bodies encoded as plain-text or quoted-printable using the UTF-8 or US-ASCII character sets. If a message is successfully parsed, the interpreter is invoked as a new process and run against the parsed message. If the message is not able to be parsed, the message is discarded.



**Figure 6:** *Quoted-Printable example captured from OpenSSL.*



**Figure 7:** *Multipart example with plain/text and text/html encoded in UTF-8 captured from OpenSSL.*

**Message Interpreting and Execution**

The bot’s interpreter loads and processes parsed messages generated by receive connectors. Each message is composed of a header and a body. The ending of the header and the beginning of the body is identified by the first empty line in a message.



**Figure 8:** *Message instructing a bot to forward a message to alias, bot1reply, instructing the bot using that alias to start powershell and execute systeminfo.*

The interpreter only interprets commands from lines in the message’s header. Lines in the body are treated as inert data. Under the current command set, the forward and write commands are the only functions that read data in the body.

The interpreter’s command set is made up of four commands: forward, start, send, and write. Forward instructs the bot to send the body of a message to a local or remote alias. Start invokes a new process. Write writes a binary file from a base64 string stored in the body of a message. Send file sends a binary file to an alias as a base64 string in a message’s body.

The forward command allows the bot to forward messages to local or remote aliases. When a forward command is invoked, a data file is referenced to resolve an alias. If the alias is matched directly to a send connector, it is a local alias and the send connector is invoked with the message to be forwarded. If the alias is matched to a remote alias, the message is encapsulated inside another “forward” message that is forwarded to the local alias linked to the remote alias.



**Figure 9:** *Examples of local and remote aliases.*

When a message is forwarded, the header is discarded leaving a decapsulated message. When messages get forwarded to remote aliases, decapsulating a message reveals another “forward” command for the interpreter at a remote bot to process.



**Figure 10:** *Forwarding to local and remote aliases.*

The start command allows the bot to start new processes on the system it resides on. The start command will accept two optional sub-commands. The arguments subcommand will supply a process with arguments. The output subcommand will redirect a process’s standard out stream to an alias as a forwarded message.



**Figure 11:** *Example of a message with a start command and subcommands of arguments and output.*

File transfers are handled by the send and write commands. The send command allows a bot to read a file on the system it resides on and transmit it as a base64 string encapsulated in a message. The message that is generated is prepended with a write command that instructs a bot receiving the message to convert the base64 string back to a byte string and write it as a file with the specified filename.



**Figure 12:** *Messages exchanged while transferring a file from one bot to another using send and write commands.*

**Transmit Connectors**

Transmit connectors are programs that connect to a web service to transmit a message. Multiple connectors for multiple protocols can be used simultaneously. Every local alias corresponds to a transmit connector. When a message is forwarded, a local alias is selected. The executable and arguments are resolved to a transmit connector and the message is transmitted.

In the case of this project, a single transmit connector has been developed to send messages to Gmail via TLS enabled SMTP. The message and Gmail account credentials are supplied to the transmit connector through arguments. The connector loads the file and assembles network credentials. If a connection is made successfully, the message is transmitted.

**Conclusion**

I’ve presented a functioning botnet that utilizes Gmail accounts to interlink bots. These bots operate behind contemporary NATs and firewalls. The botnet is organized into a de-centralized message-switched network eliminating the need for a C&C.

The main focuses of development for this project are:

1. Developing a receive connector capable of connecting to a Gmail account and processing messages in the account’s mailbox.
2. Developing a rudimentary message interpreter capable of starting processes, sending and receiving files, and forwarding messages to local and remote aliases via transmit connectors.
3. Developing a transmit connector capable of sending a message to a Gmail account’s mailbox.

The application of this botnet is intended for remote surveillance and intelligence gathering. Bots are deployed across devices to cooperatively form a dark-network that operates on bot-to-bot message forwarding and receiving. The local resources and input/output functionality of a participating device will be available to the entire botnet.

By utilizing commonly used web services for communication, messages can be passed indiscriminately from one bot to another. Since this botnet is decentralized, traffic will not have a convergence point from which to identify a C&C or other point of control.

It is my attention to continue development to expand the bot’s functionality. The current implementation is limited. Further development will be done in three categories.

First, implementing a message routing protocol will be essential for running a large botnet. Bots will be able to advertise themselves as remote aliases and distribute remote aliases of other bots. If a bot is dropped from a botnet, the botnet can re-converge using remote aliases of surrounding bots. If an alias is orphaned, the alias can be reassigned to another bot in the botnet.

Second, developing send and receive connectors that access Gmail through webmail. This will further conceal botnet traffic by facilitating message forwarding over https. Bot-to-bot message forwarding will appear as ordinary browser activity to Gmail.

Third, developing bot implementations that run on the Android OS and Linux will allow the bots to be deployed to mobile devices.

**Disclaimer**

I must point out that all development and testing has been done on private virtual machines using several private Gmail accounts. This botnet has NOT been deployed to anyone’s devices without their knowledge!