Proof of Concept: Distributed Denial of Service Phone Line Attack

Landon Clipp

July 2017

www.GitHub.com/LandonTClipp/phoneDDoS

Abstract

A Denial of Service (DoS) attack is a specific type of cyber-attack that seeks to disrupt a victim's electronic system (most commonly being internet servers) by means of sending a large volume of meaningless traffic to the target server. This erroneous traffic serves to clog up resources on the target site as the system attempts to respond to all of these queries. This document outlines one such attack that may be performed on a target phone line, via the "distributed" flavor of the DoS attack (DDoS), the tools necessary to perform it, as well as common sense practices that can be used to protect the attacker's identity.

Disclaimer

The contents herein detail activities that can be used for highly illegal purposes. Performing the attacks outlined in this document on unwitting or otherwise non-consenting parties is in violation of many state and/or federal laws in most jurisdictions. The author of this document is in no way responsible for the actions of anyone motivated or informed by this document. The author of this article does not condone any activity in violation of any laws, either foreign or domestic. Consent from the target party is always required prior to the initialization of these attacks. This document only serves as an educational resource as to the type of attack vectors out there, and is aimed at companies or institutions interested in performing internal tests on their phone systems to determine, evaluate, and correct any security vulnerabilities that may be discovered as a result of these tests.

Chapter 1

Introduction

Distributed Denial of Service attacks have long been a notorious and pernicious attack vector that has been used to either disrupt or completely disable any type of electronic system capable of receiving and serving queries from the outside world. These attacks are described as "Distributed" because they involve multiple slave units all sending erroneous requests to the target server. The behavior of the slave units is orchestrated in a hierarchical fashion by a central, master unit. The most common manifestation of a DDoS attack is seen in configurations that have many compromised computers acting as slaves to the attacker's computer. These compromised computers often generate a nearly imperceptible amount internet traffic so as not to raise the suspicions of their owner, but the collective effort of all these slaves can result in an enormous amount of traffic. Fending off this attack is especially difficult because it can be hard to distinguish the erroneous traffic from the legitimate traffic.

This idea can also be applied to phone systems. Every connected phone acts as an endpoint server, capable of receiving requests and deciding either to respond or to ignore the request. If the phone system receives too many calls at once, it can become overwhelmed, thus being unable to service legitimate phone calls. In certain circumstances, this can also prevent any outgoing calls as well.

Because Twilio already provides extensive documentation on how to set up your programming environment, those steps will not be repeated here in this document. Rather, only a high-level description of what tools and methods were used will be detailed.

1.0.1 Tools and the Attack Environment

Telemarketers have long used existing third-party services to automate outgoing phone calls to potentially hundreds, thousands, even hundreds of thousands of phone numbers. Although telemarketers may be annoying, the tools they use are not illegal in any way. Much like one may use a baseball bat for legitimate sport, you can also use it for very nefarious purposes.

Twilio The Application Programming Interface (API) used in this implementation of the attack is Twilio. Twilio provides a range of automated phone services and supports many different APIs for various programming languages. Using Twilio, one can buy many different phone numbers (often for a very cheap price) with zip codes distributed across the United States. They provide extensive documentation for their API, as well as tutorials on how to

set up an automated phone system. Their Python API was used in this implementation, however they also provide support for PHP, Node, Ruby, Java, and C# (as well as the Twilio Markup Language (TwiML) primitive calls, of which the aforementioned languages are wrappers over. TwiML is simply a form of XML). If one already has access to many different phone numbers, Twilio provides support for adding phone numbers it can verify that you own for use in your automated phone system (you do not necessarily need to buy numbers from them). They also provide an automated voice system where pre-written messages can be spoken into the phone line by a machine. It is this service that will be used to distribute the attack.

Flask As described in the Twilio Python tutorials, outgoing phone calls made using the Twilio service query a specified public web address that contains all of the instructions written in your phone script. The TwiML code generated by the script, parsed by the Twilio API, has to be publicly available on the internet. Flask is a Python utility that provides a simple website server on your local computer. This Flask server responds to requests sent to a specified port on your computer (the default being port 5000, but this can be configured). All Flask needs to operate the server is the name of your script containing Twilio API calls. In all, Flask can be downloaded and run using 3 commands on the Command Line Interface (CLI). How to do this is documented online on the Twilio tutorials, so the finer details on how to set it up will not be discussed.

ngrok Having a server running on a local computer is not enough to expose that server to the public internet. There are multiple tools that can be used that automate the process of exposing a server to the internet, but the one used in this implementation is ngrok. ngrok by itself is a free utility that will generate a unique URL you can use to point external services such as URL to your local server. ngrok can also be configured to send requests to a specified port on your computer. Obviously, the port you specify must be the same one you are using for your Flask server. If not using the paid version of ngrok, each time the program is run it will generate a random unique URL. This creates a little bit of a headache because you have to update your Twilio scripts to contain the new URL each time ngrok is started. Fortunately for outgoing calls, this is not a huge issue because the URL is only changed once inside a single script, but for incoming calls to your numbers, the URL has to be manually changed on the online Twilio website for every individual number (an arduous task).

If one was to buy the paid version of ngrok, you could request a specific URL each time you start the utility, thus removing the need to update the URL each time it is started. However as mentioned before, for outgoing calls, this is really only a minor inconvenience. If you want to support incoming calls for a large amount of phone numbers, it may be worth the investment.

Linux Although this is a purely personal choice, in the author's experience Linux is much easier to use for programming than Windows because it is much more well-supported by third party vendors and introduces less OS-level restriction. Another good alternative to Linux would be OSX as it provides a Unix-like environment. However, the reader of this document may certainly choose whichever operating system they'd like.

Python Another design choice is which programming language to use. Python was chosen due to the author's familiarity with it, its relative ease of use, and the wide support Python provides for virtual environments and package installers. The author has consistently had good experiences with installing Python virtual environments because the process is so easy that "even a caveman could do it." The only issue one may have in setting up the virtual environment as described in the Twilio tutorials is external library dependencies. If you run into issues with package installation failing because some external library is not installed on your system, it is often useful to copy-paste the error messages you are getting into Google to determine how to install those libraries. In this case, both Google and some good old-fashioned persistence are your friends. The author ran into problems during the Twilio installation with the OpenSSL library not being installed. Luckily, there is rarely a problem you can encounter with computers that has not already been documented and solved online.

Readers must always remember that when using a Python virtual environment, you must always activate your virtual environment via the "source activate" command before attempting to run Twilio Python scripts!

Chapter 2

Execution

The most complicated portion of this attack is setting up the programming environment. There are two main Twilio scripts that need to be written: the first one contains the commands Twilio will use once the call has been connected. These commands can range from "pause 5 seconds", "say these words", "hang up", or any command that Twilio supports. It is this script that generates the TwiML code that is ultimately the source for your local server. The second script controls how Twilio dials the phone numbers. For instance, parameters such as which number to call, from what number to call from, whether to record the call, what URL to use for the commands (the URL your ngrok utility gives you), and many others can be specified. This script can also be written to contain any arbitrary Python code so that, for instance, one could loop the Twilio dial command so that calls are placed every n number of seconds.

The GitHub repository listed on the title page of this document shows one possible example of these two scripts. These scripts are incredibly simple to write, and users can find many other examples on the Twilio website. Here is a sample workflow for initializing and executing the script:

1. Initialize the Python virtual environment (after downloading) by sourcing the *activate* file in the virtual environment's bin/ directory:

   source activate

   or equivalently:

   . activate

2. Point the FLASK_APP environment variable to the Twilio command script. The command file on GitHub is named $callResponse1.py$. This environment variable contains the relative or absolute path of the script.

   export FLASK_APP=./callResponse1.py

3. Run Flask with the default configurations:

   flask run

   Note that you can Google how to run Flask with a different port number if desired. At this point, you will want to open up a separate terminal as Flask will take over your current shell if you don't run it as a subshell. If you don't know what that means, don't worry about it.

4. Run the ngrok executable:

   ./ngrok

   Be sure that if you manually set the Flask port number, you will need to tell ngrok to use that port instead of the default of 5000. Otherwise, you can just run ngrok with no arguments. At this point, you'll want to again open up a separate terminal as ngrok will take over this one.

5. Update the numberList Python list in your dialing script to contain the numbers you have verified with Twilio.

6. Change the URL in your dialing script to be what ngrok has provided for you. Also add the phone number you want to attack.

7. Source the Python virtual environment again (if you followed the instructions, you are in a different terminal and will need to redo this step).

8. Go to your internet browser and enter the URL given to you by ngrok. View the page source code. In Chrome, you do this by right clicking and selecting "View Page Source." If you see some TwiML code with your written responses, your server is working normally.

9. If the previous step was successful, you can now execute your dialing script.

Hopefully, the documentation in the provided example scripts will make it easy enough to see what parts of the code perform what functions. Once you understand this, you can modify the scripts to suit your own needs.

Chapter 3

Precautions and Conclusion

As should be obvious by now, this attack is made significantly easy by using Twilio. However, people concerned with protecting their anonymity in this situation need to take extra precautions to prevent any personally-identifying information being stored when signing up for Twilio services. There are a few things that can be done for this. To sign up for Twilio, you need to provide an email address, a name, and also purchase minutes and/or phone numbers. One can utilize services such as temp-mail.org to create a disposable email address to sign up for Twilio that self-destructs after a certain amount of time. Or, one may also just create a random email address using any of the standard email providers. A personal email address should never be used, as this can be traced back to your identity.

To prevent a paper trail leading back to your bank account, pre-paid debit cards can used to purchase Twilio credits. These cards can be purchased from various brick-and-mortar stores. Care would need to be taken to use cash to pay for these cards, not any electronic form of payment like credit cards or NFC chips which are easily traceable.

An extra measure of security, if one is especially meticulous with their anonymity, would be to always use some sort of encryption protocol like a Virtual Private Network (VPN), or even more preferably the Tor network, when accessing the Twilio API or any related online services like email. One of the potential dangerous with using a VPN service arises from the fact that the highest quality providers often require a subscription, creating the potential for a paper trail leading back to your identity. It is because of this fact that Tor is the best choice as it is free and well supported. Free VPN servers are often unreliable and slow, however it still may be worth considering.

When initializing an attack on a phone system, there are multiple countermeasures that astute administrators can use. One of the most obvious countermeasures is to simply block the nefarious phone numbers. The biggest limitation of this implementation is that there is a manual-intensive process of buying phone numbers from Twilio, configuring them online, and then adding the numbers to your dialing script. This may not be an issue for attackers who are extremely motivated, but it certainly creates an upper limit to how parallel one can make the attack.

Another mode of attack is number spoofing. Twilio does not provide any spoofing capabilities, however there are many services that do provide this functionality, often VoIP companies advertising as "wholesale VoIP." Spoofing is largely ideal because the programmer can spoof the call to be coming from any conceivable phone number, thus making the process of blocking the offending phone numbers nearly impossible.

However, even this has its limitations. There are services out there such as TrapCall than can "unspoof" phone numbers, and they often do not cost that much money. These services can unveil the spoofed or restricted number to determine which phone number actually initiated the call. The best solution is most likely to pair number spoofing with the wholesale purchase of phone numbers so that even if the number is unmasked, there is still little information leading back to your identity.

Conclusion It needs to be reiterated that the author of this article does not condone any illegal activities and is not liable for any attacks carried out using this implementation, nor for any trouble one might get themselves into. This document serves as an informative tool on how easy it is for phone systems to be attacked and also provides a way for network administrators to perform attacks on their own systems to test for vulnerabilities. Readers of this document are responsible for their own actions and should always exercise good judgment.