ASGARD: A Policy Modeling System for Large-Scale Network Attacks

CyberGreen Institute

August 15, 2018

1 Introduction

This document is an introduction and overview to the ASGARD policy modeling system. ASGARD enables analysts, researchers and network defenders to evaluate the impact of *policy decisions* on an attacker's ability to successfully execute an attack. In this context, we define policy as decisions that affect the behavior of large segments of the Internet, such as ASN's, ISP's or countries.

ASGARD is designed to explore the impact of policy decisions and inform the development of effective network defenses at the start point, that is, where the bots used for DDoS, spam and other attacks reside. Historically, security research has focused on defenses at the end point, that is the target of an attack. For many attacks, such as DDoS, end point defenses have limited impact because attackers have an overwhelming supply of resources. Conversely, work on developing start point defenses are often technically simpler to implement, but require convincing a skeptical audience that they will be effective.

We, the ASGARD designers, believe that this resistance is due to a lack of understanding of the *impact* of policy decisions. Network providers makes security decisions rationally, and given the constant slate of current threats, there is minimal benefit in taking the burden of enforcing a globally responsible policy that provides them no local benefit. The net result of this is well-understood; reflection attacks have been a threat for over a decade, despite the fact that the reconfiguration required to reduce this threat is minimal. ASGARD provides the capacity to model the impact of global policy decisions, such as host reconfiguration, inventory management and egress control.

ASGARD evaluates impact via game-based Monte Carlo simulations conducted at two levels. The first (*low-level*) simulation represents a DDoS attack conducted by a botnet. The low level simulation uses a game-based model to evaluate attacker success or failure: the attacker succeeds when they overwhelm the defender's capacity to absorb the traffic which the attacker sends. The second (*high-level*) simulation runs multiple low-level simulations using randomly selected botnets; in this way, the high level simulation can evaluate the potential impact of policy decisions. Low-level simulations provide technical fidelity by modeling attacks using realistic numbers; high level simulations evaluate the impact of policy responses.

By splitting the system between low-level and high-level simulations, ASGARD analysts can run simulations of arbitrary complexity. In particular, ASGARD can simulate multistage attacks such as reflection, as well as adaptive defenses such as scrubbing and stateful policies like rate limiting. Future simulations will include dynamic reconfiguration and richer network models representing current Internet topology.

The remainder of this manual is structured as follows. §2 provides a technical overview and a mechanical description of the ASGARD system. On completion of this section, the reader will understand how ASGARD works and the basic concepts behind the simulator. §3 discusses the construction and evaluation of experiments. §A provides the manual page and operation description. §B provides descriptions of the objects in the system.

1.1 History and Versioning

Table 1 contains a revision history for ASGARD ; this table is updated every time a minor revision is posted. ASGARD releases follow a major.minor.fix format, where the major revision refers to significant updates in the system's functionality or usage, minor revisions reflect new features, and fixes refer to patches or updates in response to identified bugs.

Questions about revisions or updates should be sent to Michael Collins, `mpcollins@cybergreen.net`.

Version	Release Date	Description
1.0.0	August 15, 2018	Initial release

Table 1: Version History Table

2 Overview and Definitions

This section is an overview of the ASGARD architecture and core technologies. ASGARD is a discrete event simulator that uses Monte Carlo technique to evaluate the impact of security policy decisions on Internet-wide attacks. To do so, ASGARD combines two levels of simulation: *low-level* and *high-level*. Low-level simulations model a single attack in depth, while high-level simulations combine multiple low-level simulations to generate a distribution of results. This section is divided as follows, §2.1 describes the basic operation of ASGARD , §2.2 provides definitions for working vocabulary §2.3 provides a walk-through of the architecture, and §2.4 describes the token exhaustion model used for evaluation.

2.1 Basic ASGARD operation

ASGARD is designed to explore the impact of policy decisions and inform the development of effective network defenses at the start point. Start point, in the context of this document, means the point of origin for an attack – the individual bots which make up a DDoS. This contrasts with endpoint defenses such as scrubbing, rate limiting or blocking in that the start point defenses are distributed across multiple locations. ASGARD analyses involves developing models of these policies, then applying them to botnets which are created through network telemetry and evaluating the impact of the policy decision.

Figure 1 is a high-level diagram of standard ASGARD usage. As this figure shows, ASGARD is executed in a series of discrete steps. First, the analyst must partition the Internet into a set of *policy domains*; a policy domain is a collection of IP addresses assumed to be under the control of one organization – the default policy domain is an autonomous system, but ASGARD can use any arbitrary collection of addresses as a policy domain. Second, the analyst creates different universes – each universe is an identically partitioned Internet with different policies applied to it. In Figure 1, the Internet is broken into four policy domains, and the analyst has chosen to evaluate different policies applied on domains 2 and 3. The third step is the Monte Carlo analysis – ASGARD executes multiple simulations using randomly selected botnets and evaluates their impact. The final step provides high-level results; a sequence of histograms show the expected damage within each policy regime and enable the analyst to estimate the impact. ASGARD evaluation determines the impact that policy has on an attacker's ability to effectively damage a target; this evaluation can be used to complement other data to make a final policy decision.

2.2 Terminology

This section defines the terminology used by ASGARD.

Attacker The entity controlling the attacks and wins when the *target* is exhausted.

Bot A single computer controlled by an *attacker*.

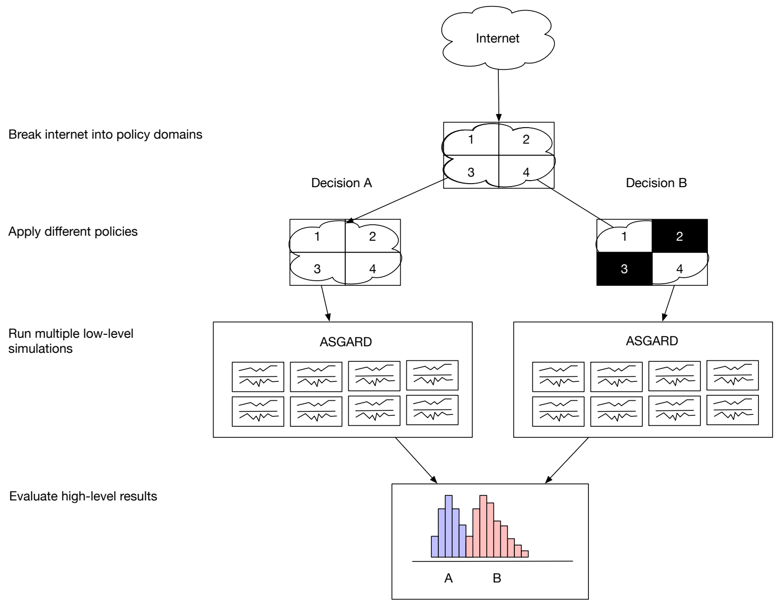
Internet
Break internet into policy domains
1
2
3
4
Decision A
Decision B
Apply different policies
Run multiple low-level simulations
ASGARD
ASGARD
Evaluate high-level results
A
B

Figure 1: Standard Usage of the ASGARD System

Bot Pool A set of *bots* which can potentially be used during an attack. In individual rounds, the *attacker* uses a subset of that *bot pool*.

Configuration The *policy regime* and *bot pool* composition decisions made before an individual *run*;

Damage In the context of the ASGARD, damage refers specifically to the resulting economic disruption caused by the attack, damage is estimated by applying a model to attacker success results.

Defender The entity which wins when the *target* is not *exhausted*.

Defense Any system which, after *policy* limits the number of tokens which can reach the *target*.

End Point The *end point* refers to the area of the attack after policy has been applied, comprising the *target* and any *defenses*

High-level Simulation Multiple *low-level simulations* conducted using randomly selected *bot pools* under a specific *policy regime*.

Layer An abstract representation of where action can take place in a DDoS attack, affecting *tokens*, *policy* and *defense*. The layer can be *link* (bandwidth exhaustion), *network* (packet exhaustion for routers) or *Service* (where the target is a service such as a web server or email server).

Low-level Simulation A game-based simulation of a DDoS attack on a single *target* using a specific *bot pool* under a specific *policy regime*.

Policy A set of behavioral constraints applied to traffic exiting a specific *policy domain*.

Policy Domain A collection of IP addresses under the control of a single entity; *policy domains* define the smallest unit at which a *policy* is applied.

Policy Regime A specific set of *policies* applied during a set of high-level or low-level simulations. Analysts evaluate the impact of different policy regimes to find the most effective ons.

Round The smallest unit of time in a *low-level simulation*; each round consists of the generation, filtering and application of a discrete set of *tokens* to the *target*.

Run A sequence of *rounds* using the same *bot pool* and *policy regime*. The *high-level simulation* executes multiple *runs*, each consisting of multiple *rounds*.

Start Point The point(s) of origin of an attack; start points consist of bots which issue traffic to targets.

Target In a *low-level simulation*, the subject of the attack by the *bots*, if the target is overwhelmed by *tokens* sent by the bots, the attacker wins.

Token The unit of impact for a DDoS attack; attacker bots generate tokens, and the target has a limited capacity to absorb tokens.

2.3 Architecture and Basic Components

Figure 2 shows the basic ASGARD architecture; as this figure shows, ASGARD is composed of the following components:

- Telemetry. This component manages the representation and analysis of network telemetry, which includes botnet populations, the decomposition of networks into autonomous systems and geolocation. CyberGreen provides telemetry data to partners, and uses several standard formats for these maps. These formats re discussed in the appendix.
- Attack Models. The attack models are representations of various attacker strategies, including reflection attacks, HTTP request floods and SYN floods.
- Policy Models. Policy models represent various control mechanisms which are applied at DDoS sources; examples of policy models include egress filtering, rate limiting and walled gardens.
- Defense Models. Defense models cover the target and defenses around the target.
- Damage Models. Damage models cover the economic damage caused by an attack on the target.
- Simulator. The simulator conducts Monte Carlo simulations to test the impact of different policy models on attacker capabilities.
- Interface. The interface presents ASGARD results to the outside world.

Analysts configure ASGARD by creating one or more experiment files. An experiment file is a YAML[1] file which specifies the parameters for an experiment; Figure 3 shows an example experimental file. The majority of ASGARD consists of code defining models for attackers, defenders and targets which can be accessed via YAML. Developers can expand the models provided by working directly with the ASGARD code.

2.4 The Token Exhaustion Model

To evaluate DDoS attacks, ASGARD uses a *token exhaustion* model. In this model, each attack is composed of one or more *tokens*; a token is observable information about the resources that an attacker is using for the attack. Formally, we define a token as a tuple of the form:

$$t = (\text{sip\_real}, \text{sip\_obs}, \text{layer}) \tag{1}$$

Where sip_real is the actual source IP address of the attacker, sip_obs is observed source IP address, and layer is the layer that the attacker is operating at. At this time the layer value is restricted to the following layers: link, network or service.

[1] `http://yaml.org`

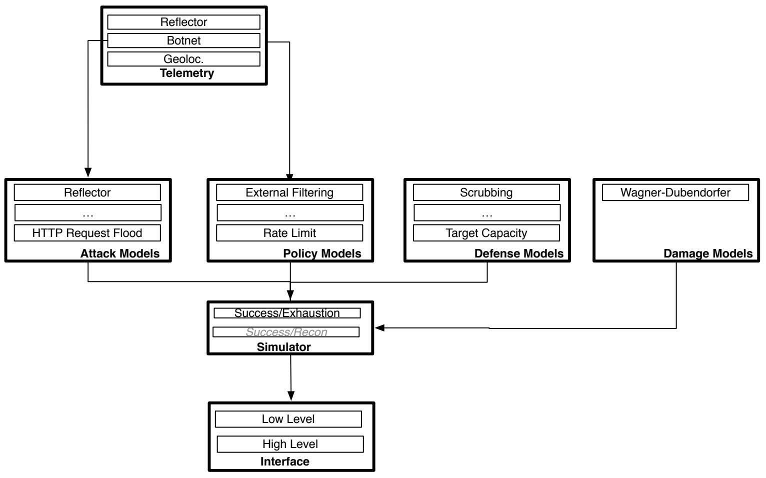

Figure 2: ASGARD Architecture

```
# Basic Spoof Attack
#
# This is a test scenario that runs through the basic features and
# provides sample output.

name: "Basic Spoof"

botnet:
  file: sampleset.set
  type: random

attacker:
  load: 60
  type: spoof

defender:
   type: scrub
   load: 25

target:
  type: basic
  capacity: 50
```

Figure 3: An Example Experiment File

During an attacker, an attacker generates a *token set* $\mathcal{T}$, which comprises all the tokens the attacker sends to the target. The *impact* on the target is expressed as follows:

$$I(\mathcal{E},\mathcal{S},\mathcal{T}) = |\mathcal{E}(\mathcal{S}(\mathcal{T}))| \tag{2}$$

Where $\mathcal{S}$ and $\mathcal{E}$ are subsets of the $\mathcal{T}$ derived by applying start point policy filtering and end point defense. Start point filtering is modeled as follows: assume a set of policies $\mathcal{P} \equiv p_1 \dots p_n$ of the general form:

$$p(t) = \begin{cases} 1 & \text{if } t \text{ passes policy} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

That is, each policy is a function that takes a token and determines whether the policy allows the token to pass (for example, an egress filtering policy passes tokens where $t.\text{sip\_real} = t.\text{sip\_obs}$). These results are used to create the set of tokens in $\mathcal{S}$, which equates to those tokens which passed all the policies implemented by the policy regime:

$$\mathcal{S}(\mathcal{T}) \equiv \{t \in \mathcal{T} | ((\prod_{p \in \mathcal{P}} p(t)) = 1)\} \tag{4}$$

Similarly, end point filtering consists of a ruleset $\mathcal{R} \equiv r_1 \dots r_n$, which are applied to the tokens at the end point. As with p, each rule is an individual function which returns 1 or 0 depending on if it passes the token.

$$r(t) = \begin{cases} 1 & \text{if } t \text{ passes rule} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

And

$$\mathcal{E}(\mathcal{T}) \equiv \{t \in \mathcal{T} | ((\prod_{r \in \mathcal{R}} r(t)) = 1)\} \tag{6}$$

The impact, I is consequently the number of tokens which passed both start point and end point filters. To evaluate whether an attacker "succeeds" or "fails" in an attack, for a given tokenset $\mathcal{T}$, we assume the defender has a *load* $L(\mathrm{d})$. We then define success as:

$$S(\mathcal{T},\mathcal{E},\mathcal{S},L(d)) = \begin{cases} 1 & \text{i}f\ I(\mathcal{E},\mathcal{S},\mathcal{T}) \geq L(d) \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

The attacker succeeds if $I(\mathcal{T}) \geq L(d)$ To prevent token exhaustion, a sequence of defensive steps *filter* traffic from the attacker to the defender.

3 Experimental Structure and Output

This section covers the construction and interpretation of ASGARD results. It is broken down as follows: 3.1 provides a high level description of the experimental process; 3.2 describes *tokens* and how to create them; 3.3 discusses the process of interpreting results.

3.1 Experiment Sequence

To evaluate different policy decisions, analysts create experiment files (as shown in Figure 3), and then conduct simulated *runs*, comparing the results of different experiments. Each run is a Monte Carlo simulation deriving values for Equation 2 using different randomly selected botnets collected from telemetry data. Figure 4 shows the mechanical process used to implement Equation 2. The sequence shown in Figure 4 takes place in a discrete package termed a *round.* Each simulated *run* consists of a sequence of *rounds*, which

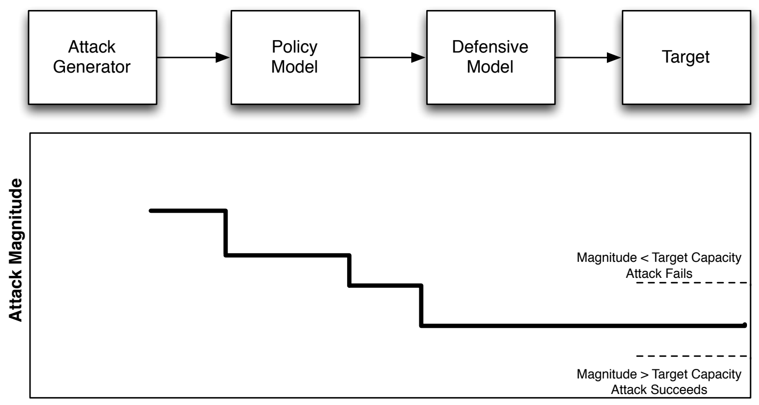

Figure 4: Filtering Sequence

represent the progression and adaptation of attackers and defenders during the attack. During a round, the following sequence of operations occurs:

1. The attack generator creates $\mathcal{T}$for the round.
2. Policy models are applied to $\mathcal{T}$, producing $\mathcal{S}(\mathcal{T})$.
    (a) If needed, policy models update their state based on the attack (see walled garden models for an example)
3. Defenses are applied to $\mathcal{S}(\mathcal{T})$, producing $\mathcal{E}(\mathcal{S}(\mathcal{T}))$.
    (a) If needed, defensive models update their state based on the attack.
4. The simulator calculates I, and compares it to $L(d)$ to determine if the attacker succeeded.
5. Damage is updated based on whether the attacker succeeded or failed.
6. The process is repeated until all rounds have completed.

Figure 5 shows an example of the output of a single run. The output shown in Figure 5 is a set of time series executed in parallel, showing the state of the simulation for different values. The values shown in this figure are:

Bots This line describes the population of bots engaged in the attack.

Atk. Succ This line shows the attacker success – values in this line will be a 1 if $I(\mathcal{T}) \geq L(d)$.

Damage This lines shows the damage incurred by the attack as described by the current damage model.

Target This line shows $|\mathcal{E}(\mathcal{S}(\mathcal{T})|$

Defense This line shows $\mathcal{S}(\mathcal{T})$ and the load on the defenses.

Note the red lines in the Bots, Target and Defense boxes; these lines represent the *loads* on the represented systems. The loads represent the maximum value that each system can manage. For bots, this refers to the maximum number of bots that the bot master has available for the attack. For the target, this refers to the maximum number of tokens the target can process. For the defense, this refers to the maximum number of rules that the defense can manage before being overloaded.

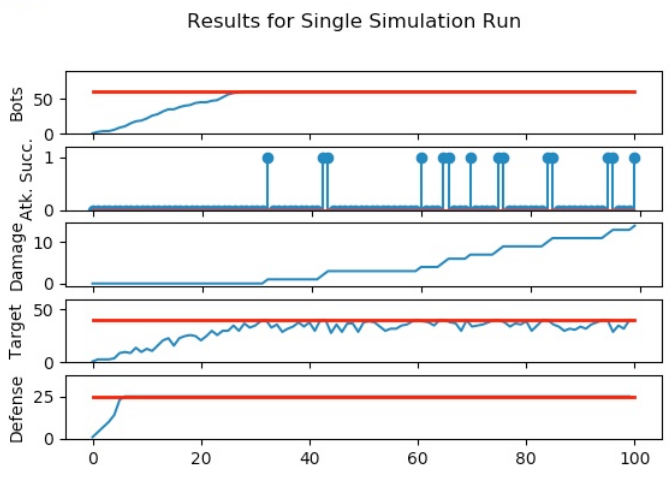

Figure 5: Example Output from a Single Run

The load lines show whether an attacker is succeeding or failing, and inform the other two values – damage and attacker success. An attack succeeds when the tokens received by the defender exceed the defender's load line; this can be seen in Figure 5 by comparing the attacker success values to the peaks of the target line.

Damage is a function of the attacker successes, and is calculated using a distinct damage model. As of this implementation, ASGARD provides a model based on the Dubendorfer damage model. To evaluate the impact of distinct policy decisions, ASGARDcompares runs with different policy environments. In this context, a single experimental *run* will consist of multiple rounds with shared state. This state includes the following:

- The composition of the botnet used by the attacker – bots used, and bots available.
- Policy choices and configuration.
- Defensive state; in this iteration, blocking rules implemented by a scrubber.

The output from the high-level simulation is a histogram, an example of which is shown in Figure 6.

Effective defensive policies will result in reduced damage; consequently, the output of Figure 6 for effective policies will distribute further left than less effective policies.

3.2 What Tokens Mean

A *token* is an abstract representation of the damage a DDoS attack causes. In order to produce a realistic simulation, the analyst must understand how to create effective tokens. This sub section discusses the process of token creation, by explaining are defined and expressed, discussing the tradeoffs in token construction, and providing historical examples of DDoS attacks.

Tokens are defined through their interaction between two attributes in the attacker (§B.1) and target (§B.3) objects. As described by Equation 7, an attack succeeds when an attacker generates more tokens then the defender can process. This leads to two key points: tokens are related to the real world by the target's load, and tokens are quantized.

A target's load represents the capacity a target has for processing *something* before being overwhelmed. The something varies on the layer that the DDoS is attacking – for example, at the link layer the load is

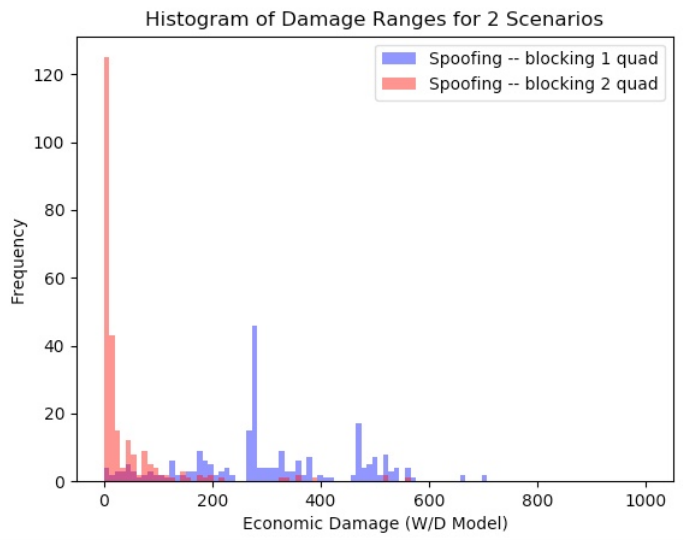

Figure 6: Example Output from Multiple runs, Showing the Impact of Different Policy Decisions

Token Size	Bot Token Range	Load	Notes
1 bit	10e5-10e7	10e9	Setting a token size this small will require processing billions of tokens a round, resulting in a very slow simulation.
1 Mibit	.1-100	10e3	Setting token size to a Mibibit (1 million bits[2] requires only a thousand tokens to model exhaustion, but results in unrealistically high traffic coming from individual hosts.
100 Kibits	1 - 1000	10e4	Setting this size results in potentially realistic attack sizes, and is computationally reasonable.

Table 2: How Token size and loads Interact

usually bandwidth and expressed in terms such as Gbps (Gigabits per second), the network layer will express load in terms of pps (packets per second), and a service layer will represent it in terms of requests per second. For example, consider an HTTP exhaustion attack – the target of such an attack will be able to process between 100 and 1000 requests per second, in which case it's reasonable to set a token to a single request, and the load of the target between 100 and 1000 tokens.

Token quantization means that the smallest unit used in an attack is a token, this is more critical as we consider higher-volume targets. For example, consider an attack on a Gigabit Ethernet connection. This connection can take 10e9 bits of traffic before being exhausted. Table 2 shows some example token sizes and loads that could be used in this case.

The supplemental spreadsheet `incident_database.xlsx` provides historical examples of incidents and potential token and load values.

3.3 Results: Understanding Success

ASGARD defines success in two ways – the low-level simulator, defines success using Equation 7. This definition of success is binary and zero-sum – either the attacker has succeeded, or the defender has succeeded, and there is no intermediary state. The high level simulator uses *damage models* to evaluate success; a damage model calculates damage over a sequence of Equation 7 outputs, one for each round of the simulation.

In the current implementation, the damage model is derived from Dubendorfer *et al.*'s work on estimating the impact of DDoS attacks, and refers to economic damage[3]. The Dubendorfer model identifies four classes of economic damage, these are:

Downtime Loss Downtime loss refers to immediate costs due to the targeted site being out of operation; this includes revenue loss from the site being unable to service customers, as well as losses due to employees being unable to fulfill their jobs. Downtime loss is modeled as a linear function of time down – each round that the attacker succeeds adds further downtime loss.

Disaster Recovery Disaster recovery loss refers to the time and effort required to bring the system online after an attack. Disaster recovery is modeled using a threshold – if the attacker is able to keep the target down for longer than a some number of rounds, disaster recovery is incurred.

Liability Liability refers to costs incurred due to inability to fulfill site obligations. Liability is incurred if the attacker wins for some minimum number of rounds.

Customer Loss Customer loss refers to costs incurred due to loss of reputation and long-term customer attrition. Customer loss is incurred if the attacker wins for some minimum number of rounds.

The damage line in Figure 5 shows the estimated damage from an attack. When comparing different policy choices, the damage is modeled as shown in Figure 6. The histograms shown in Figure 6 show the distribution of different damage results for different policy regimes. A policy regime is more effective if, given all other things being equal, the aggregate damage caused by the attack is lower.

When using ASGARD for policy analysis, analysts must be aware of the key assumptions made behind the system. These assumptions derive from a basic operating principle: that DDoS attacks are *not subtle*. From that, we assume that variations in detection or topology are not germane, and make assumptions that benefit the defender.

In particular, ASGARD assumes the attacker is going to overwhelm the target sufficiently that normal user traffic does not contribute to the simulation. For this reason, the current implementation of ASGARD does not include a user traffic generator; this is implicitly handled via loads. In addition, the current implementation ASGARD does not address the impact of network topology on traffic, since the policy decisions that ASGARD simulates would effectively *reduce* the amount of attack traffic coming from individual networks.

We note in passing that these assumptions affect the *current* implementation of ASGARD ; future versions will include more complex simulations.

4 Conclusions

This document has introduced ASGARD, a system for evaluating the impact of policy decisions on distributed attacks. Using ASGARD, a policy analyst can build models for different defensive policies and determine how they will affect an attacker's ability to damage targets.

The strength of the two-level simulator approach is that it enables analysts to expand the simulator to accomodate different forms of attacks and defenses. The next generation of ASGARD will develop higher-fidelity low-level simulations by combining traffic telemetry and adaptive defenses.

A Manpages

The manpage shown in Figure 7 is taken directly from the command line. ASGARD is invoked by running the `asgard.py` script from Python. ASGARD uses Python 3 language features; running the tool with Python 2 will result in syntax errors and early termination.

[3]Dubendorfer, T. *et al.*, *An Economic Damage Model for Large-Scale Internet Attacks*, in 2004 IEEE Workshops on Enabling Technologies

```
usage: asgard.py [-h] [--onerun] [--compare] [--round-count ROUND_COUNT]
                 [--run-count RUN_COUNT] [--output OUTPUT_FORMAT]
                 scenario [scenario ...]

Process Simulations

positional arguments:
  scenario              Scenario file

optional arguments:
  -h, --help            show this help message and exit
  --low-level            Run low level simulation
  --high-level           Run high level simulation
  --round-count ROUND_COUNT
                        Number of rounds per run
  --run-count RUN_COUNT
                        Number of individual runs per experiment
  --output OUTPUT_FORMAT
                        Preferred output format, can be: ui, or csv
usage: asgard.py [-h] [--low-level] [--high-level] [--round-count ROUND_COUNT]
                 [--run-count RUN_COUNT] [--output OUTPUT_FORMAT]
                 [--beep BEEP]
                 scenario [scenario ...]

Process Simulations

positional arguments:
  scenario              Scenario file

optional arguments:
  -h, --help            show this help message and exit
  --low-level           Low level simulation
  --high-level          Compare scenarios
  --round-count ROUND_COUNT
                        Number of rounds per run
  --run-count RUN_COUNT
                        Number of individual runs per experiment
  --output OUTPUT_FORMAT
                        Preferred output format, can be: json, csv, text or ui
  --beep BEEP           Outputs a period to stderr every specified number of
                        rounds to indicate operation
```

Figure 7: ASGARD Manpage, Taken from Command Line

The basic invocation of ASGARD is of the form `asgard.py -low-level scenario.yml`; this will run a low level simulation as configured by the `scenario.yml` file (the distribution includes multiple scenario files for testing and training purposes) and display the result to the UI. The operator can, alternatively, run a high level simulation by specifying `-high-level` instead, this will generate histograms.

The `asgard.py` tool has no upper limit on the number of scenarios specified. In the case of low level simulations, this will result in progressive displays on screen – that is, a low-level simulation will be run for the first file, the results shown, then when that window is closed, the next scenario will run. In the case of a high-level simulation, this specifies the number of histograms plotted on the output.

The number of simulations processed is specified by the `round-count` and `run-count` arguments. `round-count` specifies the number of rounds in a run and defaults to 20 rounds. `run-count` specifies the number of runs for a high-level simulation and defaults to 20 runs. Note that the `run-count` argument is only relevant for high-level simulations.

The `-output` argument enables different output formats. `ui`, the default, produces the plots seen in this document. The `csv` option will instead dump comma-separated value format data to screen, which can then be processed via Excel, R or any other data analysis tool.

B Object Model

The basic ASGARD object hierarchy is shown in Figure 8; as this figure shows, ASGARD consists of a base set of classes: attackers, botnetselectors, targets, policymaps, policies and defenders. Each of these classes contains one or more implementations, which are accessible by configuring the relevant feature in an experiment file.

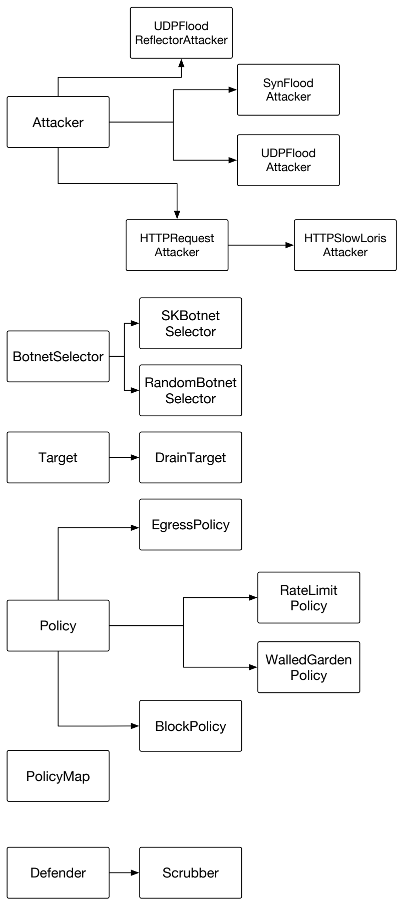

Figure 8: ASGARD Basic Simulation Object Hierarchy

B.1 Attacker Objects

type	base|spoof|udpflood|reflector| httpflood	Attacker type
load	int	Maximum botnet size
delta_min	int	Minimum change in botnet size per round
delta_max	int	Maximum change in botnet size per round
init_size	int	Initial botnet size
tok_min	int	Minimum number of tokens generated per bot per round
tok_max	int	Maximum number of tokens generated per bot per round
spoof	bool	Whether or not the attack is spoofed

Table 3: Base Configuration Values for Attacker Objects

Table 3 shows the possible attributes for a attacker configuration. Attack objects are specified as one of the following types:

- **base** Baseline attacker; sends TCP packets to the target without spoofing.
- **udpflood**. UDP flood attacker; this sends UDP packets without spoofing.
- **spoof**. Spoofed TCP flood.
- **httpflood**. HTTP attacker; this sends instantaneous HTTP request sessions. It cannot be used with spoofing.
- **reflector**. Reflector attacker; this sends UDP packets to a reflector set which are then sent to the target. These attacks are spoofed.

The other arguments specify attributes of the attack. The *load* value specifies the maximum size of the botnet that the attacker can use; if the *load* value is larger than the supplied botnet, then the botnet's size will be used. The *init_ size*, *delta_ min* and *delta_ max* values are used to specify the size of the botnet each round; in the first round, the attacker selects *init_ size* bots, then adds a random value between *delta_ min* and *delta_ max* each round until the active size is equal to *load*. Formally, given a round r, the number of active bots in r is:

$$B_r = B_{r-1} + R(\text{delta\_min}, \text{delta\_max}) \tag{8}$$

$$B_0 \equiv \text{init\_size} \tag{9}$$

The *tok_ min* and *tok_ max* values determine the range of token values generated by an active token. The total tokens sent by the host are evaluated as:

$$\mathcal{T}(r) = \sum_{i=1\ldots B_r} R(\text{tok\_min}, \text{tok\_max}) \tag{10}$$

```
1.3.9.11
2.18.3.17
211.14.9.44
```

Figure 9: A Simple IP list for Botnet Selection

B.2 Selector Objects

type	silk|random|text	Selector type
file	string	Filename
load	int	maximum entries

Table 4: Base Configuration Values for Botnet Selector Objects

The *Selector* object is used to select bots for an attacker. Selectors come in three types:

- **silk** The file is in SiLK IPSet format.
- **random** The file is randomly generated.
- **text** The file is a text file of IPv4 Addresses.

ASGARD currently supports two file formats for botnet selectors: SiLK format and text format (the random "format" uses randomly generated addresses across IPv4 space). SiLK format, in this context, refers to the SiLK IPset format, a compact binary representation of IPv4 addresses – this format is only usable if the SiLK tool suite is installed [4]. Data in SiLK format must be generated using the SiLK `rwsetbuild` or `rwset` command. Text format consists of IPv4 addresses expressed in standard "dotted quad" format (e.g., '151.101.64.144'). Figure 9 shows an example of this format.

The *load* argument implements an upper limit to the number of unique bots that the selector will provide in one call. If the load value is greater than the number of hosts available in a file, then the population of the file will be used. In the case of a 'random' selector, the load value is effectively meaningless.

[4] More information on SiLK is available at `https://tools.netsa.cert.org/`

B.3 Target Objects

type	basic|drain	Target type
load	int	Maximum target capacity
timeout	int	Rate at which tokens are reclaimed

Table 5: Base Configuration Values for Target Objects

Target objects model the target's ability to receive and process damage. The current implementation provides two target model types:

- **basic** Base target type; only processes the tokens received that round.
- **drain** Drain target type; keeps the tokens received in a round for future rounds.

The *load* value refers to the number of tokens the target can manage in a round. If the target receives more than *load* tokens, then the attacker succeeds; the the target receives less than *load*, then the attacker fails.

The default (i.e., *basic*) type target 'clears out' tokens each round; that is, the target will assume that the only tokens mattering are the tokens generated in a round. *Drain* type targets are used to model attacks such as slowloris, which maintain longer connections. Drain targets use the *timeout* argument to specify how long a token lasts in the load; the argument is an integer for the number of rounds before the token is cleaned out.

```
# An example policy map
# Lines beginning with a hashmark are treated as comments and ignored
# during loading.
# Addresses are specified as CIDR blocks followed by the domain ID
  128.2.0.0/24 1
# Multiple blocks can belong to the same domain, just specify the
# ID again.
  38.0.17.0/22 1
# Blocks can be any size from /8 to /32
  218.9.0.0/15 2
# Domains do not have to be assigned sequentially
  99.3.4.17/32 1
```

Figure 10: An Example Policy Map

B.4 Policy Objects

type	basic|egress|block|rate|garden|remediate	Policy type
load	int	Maximum load size
mapfile	string	Pointer to map file name
domains	int list	domains affected by map
chance	int	probability (out of 1000) that a host was remediated

Table 6: Base Configuration Values for Policies

Policy objects model the impact of network policy on attacker behavior. All policy objects work by applying their policy across a *domain*; in the context of ASGARD , a policy domain is a collection of IP addresses under the control of a single entity (meaning that the entity has the ability to effectively implement policy within that domain). Policy domains are described using *policy maps*, which are tables listing a domain and its constituent IP addresses. An example policy map is shown in Figure 10.

There are five basic policy types, these are:

- **base** The base policy type does nothing. That is:

$$p_{\text{base}}(t) \equiv 1 \tag{11}$$

- **egress** The egress policy type blocks spoofed outgoing traffic. That is:

$$p_{\text{egress}}(t) = \begin{cases} 1 & \text{if } t.\text{sip\_real} = t.\text{sip\_obs} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

- **rate** The rate limit policy limits the maximum number of tokens that can exit a network.
- **garden** The walled garden policy stops a bot generating tokens after its initial attack.
- **block** The block policy stops a bot from communicating.
- **remediate** The remediate policy expresses a probability (using the optional argument *chance* that a host has been patched.

The *load* argument is currently only used with the rate limit policy and specifies the maximum number of tokens a domain can issue. The *chance* argument is only used with the remediate policy and expresses a probability out of 1000 (*e.g.*, setting chance to 500 equates to a one out of two chance) that a host is remediated. Remediated hosts will not generate attack traffic.

B.5 Defender Objects

type	base|scrub	Defender type
load	int	Maximum load size

Table 7: Base Configuration Values for Defender Objects

Defender objects specify endpoint defense capabilities. In the current implementation, there are two types:

- **base** The base defense does nothing.
- **scrub** A scrub defense blocks IP addresses.

The *scrub* defender keeps a blacklist of IP addresses; when the defender receives a token, it compares that token's source address [5] with the blacklist. If the address is on the blacklist, the attack is dropped. If the address is not on the blacklist, it is added to the blacklist. The defender can manage up to *load* addresses; after exhausting its blacklist it lets tokens through.

[5] note that this is the published source address and, if the attacker spoofs, will be spoofed