SAND2017-12873C

Enhanced Dynamic Cyber Zone Defense

Marci McBride
Cyber Security Technologies
Sandia National Laboratories
Albuquerque, NM
Email: mmcbri@sandia.gov

Robert Mitchell
Cyber Security Technologies
Sandia National Laboratories
Albuquerque, NM
Email: rrmitch@sandia.gov

***Abstract*—Information security is a top priority in government and industry because high consequence cyber incidents continue with regularity. The blue teamers that protect cyber systems cannot stop or even know about all these incidents, so they must take measures to tolerate these incursions in addition to preventing and detecting them. We propose dynamically compartmentalizing subject networks into collaboration zones and limiting the communication between these zones. In this article, we demonstrate this technique's effect on the attacker and the defender for various parameter settings using discrete-time simulation. Based on our results, we conclude that dynamic cyber zone defense is a viable intrusion tolerance technique and should be considered for technology transfer.**

***Index Terms*—moving target defense; intrusion tolerance; security; algorithm; simulation**

I. Introduction

Government agencies and corporate enterprises make information security a top priority because of cyberspace's hostility. The Dyn domain name system (DNS) distributed denial of service (DDoS), San Francisco Municipal Transportation Agency ransom, WannaCry pandemic, Petya attack and Equifax personally identifiable information (PII) spill demonstrate that cyber events are not decreasing in frequency or impact.

Cyber security groups cannot stop or even know about all these events, so they must provide a defense in depth that includes an intrusion tolerance layer. We propose dynamically dividing subject networks into zones based on business needs and limiting the traffic that flows between these zones. We demonstrate this technique's effect on the attacker and the defender for various parameter settings using discrete-time simulation. This approach can be considered a moving target defense (MTD).

Our threat model comprises four key points: First, we assume that we face a sophisticated state or criminally-sponsored adversary. Next, we assume that an insider threat is present. Third, we assume that this insider must move laterally to their final objective. Finally, we assume that they must reach back to their command and control (C2) node.

Now we visualize the dynamic cyber zone defense use cases in three figures.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

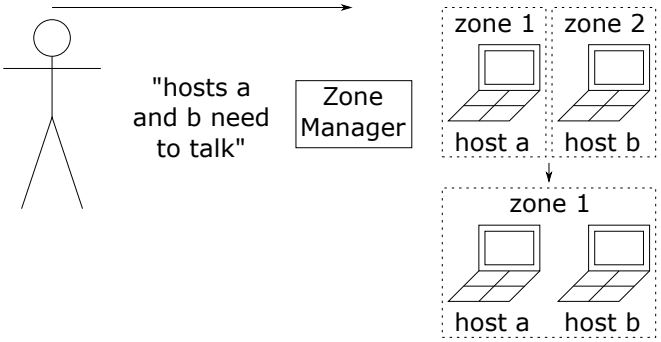

Fig. 1. Human-triggered rezoning.

Figure 1 shows a human identifying a business need for hosts a and b to communicate freely. A host can be any endpoint on the subject network: for example, a workstation, server, printer or scientific device. This human will submit a request to the Zone Manager (ZM). The ZM collects these requests across the enterprise and fulfills them. The ZM can fulfill these requests on an event-driven basis: for example, fulfilling every request as it arrives. Alternatively, the ZM can do this periodically: for example, every night at midnight. To fulfill each request, the ZM assigns the two hosts to the same zone. Generally speaking, a zone is a network partition; layer 2 (e.g., passive optical network (PON)) and layer 3 (e.g., Internet protocol (IP)) are the most obvious places to effect the zone boundaries. However, this work is not specific to any layer or protocol.

Figure 2 shows the ZM enforcing one facet of security policy: limiting the number of zones a host belongs to. This figure shows host c's zone membership and includes a timestamp indicating the last time host c used each zone. The security policy limits each host to two zones, so the ZM reduces host c's zone count. Specifically, the ZM removes host c from zone 4 because host c used this zone least recently. Like the previous use case, the ZM can perform this function asynchronously or periodically.

Figure 3 shows the ZM enforcing a different aspect of security policy. In this case, it limits the number of hosts that belong to a zone. Specifically, the ZM reduces zone 6's membership to three hosts; it removes host f from zone 6 because host f used this zone least recently. Like the previous use cases, the ZM can do this asynchronously or periodically.

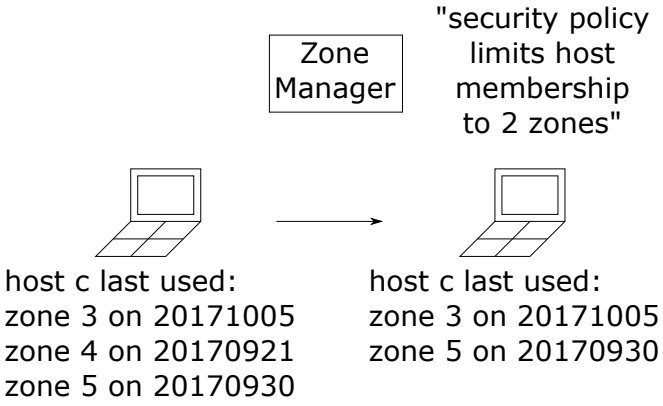

Fig. 2. Policy-based host devaluation.

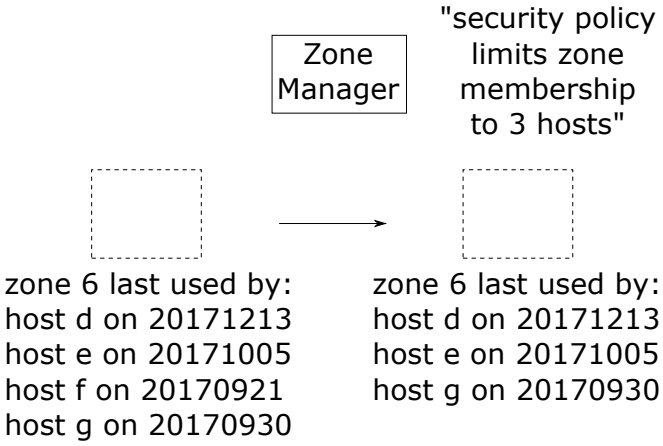

Fig. 3. Policy-based zone devaluation.

The rest of this paper is organized as follows: First, Section II surveys the state of the art for this topic. Next, Section III proposes an algorithm to solve the problem at hand. Third, Section IV describes the discrete-time simulation that we instrumented to study this technique. Next, Section V presents numerical data and figures that quantifies and visualize the performance of this implementation of dynamic cyber zone defense. Finally, Section VI concludes the article by summarizing our contributions in the study and identifying future lines of investigation.

II. Literature Search

In this section, we survey the contemporary work related to dynamic cyber zone defense.

Mitchell, et al. wrote three papers which propose increasingly intricate closed-form mathematical models that predict how static cyber zone defense affects attackers in [5], [4], [6]. Our investigation studies dynamic cyber zone defense and researches the impact to the defender as well as the attacker.

Earlier, we wrote an article proposing dynamic cyber zone defense [2]. However, this work did not include any attacker-facing metrics; all the measurements related to the impact this technique would have on the defender. Also, this earlier approach arbitrarily removed humans and hosts from zones. In the present work, we provide two attacker-facing metrics and use business-related criteria for deciding what hosts to remove from zones.

Algorithm 1 Request fulfillment.

```
1:  function FULFILL_REQUESTS(r, Z)
2:      for r ∈ r do
3:          if r_0 = r_1 then
4:              continue
5:          else if ∅ ≠ r_0.Z ∩ r_1.Z then
6:              continue
7:          else if ∅ ≠ r_0.Z || ∅ ≠ r_1.Z then
8:              z ← find_smallest_zone(r_0, r_1)
9:              if r_0 ∈ z then
10:                 r_1.Z ← r_1.Z ∪ z
11:                 z.H ← z.H ∪ r_1
12:             else
13:                 r_0.Z ← r_0.Z ∪ z
14:                 z.H ← z.H ∪ r_0
15:             end if
16:         else
17:             z ← find_smallest_zone(Z)
18:             r_0.Z ← r_0.Z ∪ z
19:             r_1.Z ← r_1.Z ∪ z
20:             z.H ← z.H ∪ r_0 ∪ r_1
21:         end if
22:         devalue_host(r_0)
23:         devalue_host(r_1)
24:     end for
25: end function
```

III. Algorithm

A. Implementation

In this section, we propose dynamic cyber zone defense algorithms and analyze their runtime complexities. In terms of notation, the boldface variables are vectors, and the capital variables are sets in Algorithms 1 through 4.

As discussed in Section I, the ZM can invoke Algorithm 1, fulfill_hosts(), periodically or on an event-driven basis. This algorithm accepts two parameters: $\mathbf{r}$ is the list of requests that the ZM has pending. Each request comprises two components; r_0 and r_1 are the two hosts for which there is a business need to be in the same zone. In terms of notation, $r_x.Z$ indicates the set of zones that r_x is a member of. The other parameter is Z which is the set of all zones that the ZM administers. When invoked, fulfill_requests() begins by iterating over the list of requests. If the two hosts that comprise the request are the same or if the two hosts are already members of the same zone, Algorithm 1 proceeds to the next request. Otherwise, if at least one of the two hosts is in a zone, fulfill_requests() finds the smallest zone that either of them belongs to and adds the other host to this smallest zone. Otherwise, Algorithm 1 finds the smallest zone overall and adds both hosts to this smallest zone. Finally, this algorithm devalues each host. One

Algorithm 2 Host devaluation.

```
1: function DEVALUE_HOST(h)
2:     while |h.Z| > mzph do
3:         z ← least_recently_used_zone(h)
4:         z.H ← z.H − h
5:         h.Z ← h.Z − z
6:     end while
7: end function
```

Algorithm 3 Zone policy enforcement.

```
1: function ENFORCE_ZONE_POLICY(Z)
2:     for z ∈ Z do
3:         devalue_zone(z)
4:     end for
5: end function
```

optimization is to devalue only the hosts added to a zone, but we trade conceptual compactness for this optimization.

During its execution, Algorithm 1 invokes Algorithm 2, devalue_host(). This algorithm accepts one parameter: h is the host to be devalued. A host that is a member of too many zones is a high value target, so we refer to the process of removing it from some zones as devaluing the host. Algorithm 2 repeats the following steps until h belongs to an acceptable number of zones: (mzph, the maximum zones per host, defines this acceptable number.) First, devalue_host() finds the zone z that h used least recently. In terms of notation, $z.H$ indicates the set of hosts that belong to z. Next, this algorithm removes h from $z.H$. Finally, Algorithm 2 removes z from $h.Z$.

As discussed in Section I, the ZM can invoke Algorithm 3, enforce_zonepolicy(), periodically or on an event-driven basis. This algorithm accepts one parameter: Z is the set of zones that the ZM administers. Algorithm 3 iterates over all the zones in Z, and it invokes devalue_zone() for each zone z.

During its execution, Algorithm 3 invokes Algorithm 4, devalue_zone(). This algorithm accepts one parameter: z is the zone to be devalued. A zone to which too many hosts belong is a high value target, so we refer to the process of removing some hosts from it as devaluing the zone. Algorithm 4 repeats the following steps until an acceptable number of hosts (mhpz, the maximum hosts per zone, defines this acceptable number) belong to z: First, devalue_zone() finds the host h that least recently used z. Next, this algorithm removes h from $z.H$. Finally, Algorithm 4 removes z from $h.Z$.

B. Analysis

Now we analyze the worst-case runtime complexity of Algorithms 1 through 4.

In the worst-case scenario, each request that fulfill_requests() processes comprises two hosts that are not a member of a zone. Hosts may not be a member of a zone if they are rarely-used servers, for example. While processing each of these worst-case requests, Algorithm 1 iterates over two host zone-sets to calculate their intersection (line 5), the set of all zones to find the smallest zone (line 17), two host zone-sets to calculate unions (lines 18 and 19) and one zone host-set to calculate a union (line 20). This algorithm will also incur the runtime of devalue_host() twice per request in the worst case.

Algorithm 4 Zone devaluation.

```
1: function DEVALUE_ZONE(z)
2:     while |z.H| > mhpz do
3:         h ← least_recently_used_host(z)
4:         z.H ← z.H − h
5:         h.Z ← h.Z − z
6:     end while
7: end function
```

In the worst case, for every zone that the host is a member of (line 2), Algorithm 2 iterates over the host zone-set once to find the least recently used zone (line 3), the zone host-set once to remove the host (line 4) and the host zone-set a second time to remove the zone (line 5).

Therefore, the worst-case runtime complexity of devalue_host() is:

$$\begin{aligned} &O(|Z|(|Z| + |H| + |Z|)) \\ &= O(2|Z|^2 + |Z||H|) \\ &= O(|Z|^2 + |Z||H|) \end{aligned}$$

This means the worst-case runtime complexity of fulfill_requests() is:

$$\begin{aligned} &O(r(2|Z| + |Z| + 2|Z| + |H| + 2(|Z|^2 + |Z||H|))) \\ &= O(r(5|Z| + |H| + 2|Z|^2 + 2|Z||H|)) \\ &= O(r(|Z| + |H| + |Z|^2 + |Z||H|)) \end{aligned}$$

Therefore, Algorithm 1 runs in quadratic time with respect to the number of zones and in linear time with respect to the number of requests and hosts.

For each zone, enforce_zone_policy() invokes devalue_zone(). In the worst case, for every host that is a member of the zone (line 2), Algorithm 4 iterates over the zone host-set once to find the host that least recently used the zone (line 3), iterates over the zone host-set a second time to remove the host (line 4) and iterates over the host zone-set once to remove the zone (line 5). Therefore, the worst-case runtime complexity of devalue_zone() is:

$$\begin{aligned} &O(|H|(|H| + |H| + |Z|)) \\ &= O(2|H|^2 + |H||Z|) \\ &= O(|H|^2 + |H||Z|) \end{aligned}$$

This means the worst-case runtime complexity of enforce_zone_policy() is:

$$\begin{aligned} &O(|Z|(|H|^2 + |H||Z|)) \\ &= O(|H|^2|Z| + |Z|^2|H|) \end{aligned}$$

Therefore, Algorithm 3 runs in quadratic time with respect to the number of zones and with respect to the number of hosts.

TABLE I
SIMULATION PARAMETERS.

Parameter Name	Description	Source	Range
p_c	probability of compromise	Simulation	0.0 - 1.0
p_r	probability of reachback	Simulation	0.0 - 1.0
d	disruptions	Simulation	0 - 40,000
p_e	probability an exploit exists	Defender	0.1 - 0.9
z	maximum zone size	Defender	25 - 500
n	network size	Defender	2048
x	number of Internet gateways	Defender	8 - 64
Φ	interzone porosity	Defender	0.1 - 0.9
λ	mean request interval	Defender	1 - 30 d

IV. SIMULATION

We began with the simple framework we used for [3], [6]. Next, we integrated our extensions to [2]. Finally, we added the enhancements we propose in this paper. The instrumentation is a SimPy [1] artifact with three classes: a Network singleton, an active ZM singleton and an active Host class that is instantiated many times. The Network singleton is passive; it serves as a container for the simulation's objects, attributes and utility methods. The ZM singleton is active; it contains implementations for the request fulfillment and policy enforcement algorithms described in Section III and invokes them once every 24 hours of simulation time. The many Host objects are active; they bill charge numbers, create zoning requests and, if they are infected, attempt to compromise other hosts. This timecard information is the criteria by which the algorithms determine the least recently used zone. Once every 24 hours of simulation time, the Host objects choose the number of charge numbers to bill and the charge numbers themselves based on a uniform distribution. The Host objects create a zoning request for the ZM based on an exponential distribution, but choose the requested host based on a uniform distribution. Infected Host objects choose a host to attempt to spread to based on a uniform distribution, and nature determines if these attempts are successful based on a uniform distribution. We make three assumptions in the simulation: there is a one-to-one correspondence between cyber zones and charge numbers, we cannot create new zones and there is a one-to-one correspondence between humans and hosts.

Table I lists the parameters that constrain our discrete-time simulation. p_c is the probability that an arbitrary host in the subject network is compromised. p_r is the probability that an arbitrary host in the subject network can reach back to its C2 node. To reach back, the host must be compromised and have visibility to a compromised externally-facing host. d is the number of disruptions hosts experience during the simulation. A disruption occurs when the ZM removes a host from a zone in the process of devaluation (cf. Algorithms 2 and 4. p_e is the probability that an arbitrary host in the subject network is exploitable. Mitchell and Sery provide a technique to estimate p_e in [4]. z is the maximum zone size. This will be lower for a highly compartmentalized network and higher for a flatter network. n is size of the entire subject network. x the number of externally-facing hosts on the subject network. This will be lower for a cyber-physical system (CPS) that is mostly populated with sensors, actuators and control units. x will be higher for an enterprise network that is mostly populated with attended commodity devices. Φ is the porosity of the zone boundaries. [4] also provides a technique to estimate Φ. λ is the mean request interval. This will be lower for highly dynamic workplaces with new collaborations forming frequently and higher for more static workplaces and industrial control systems (ICSs).

V. RESULTS

This section visualizes the numerical results the discrete-time simulation produced in Figures 4 through 11. We also interpret the trends these figures show.

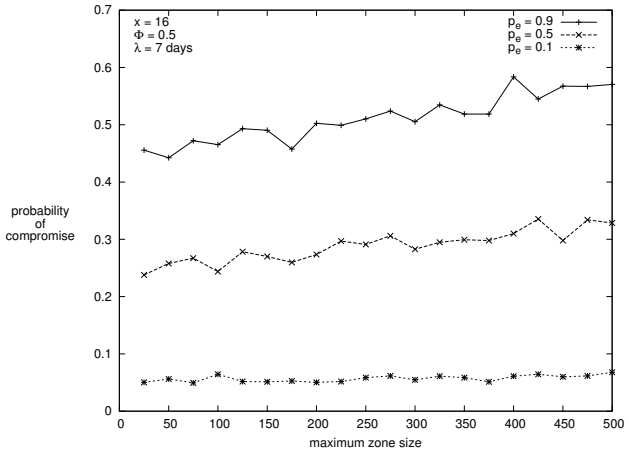

Fig. 4. Probability of compromise versus zone size and probability of exploitability.

Figure 4 shows the impact of the maximum zone size and the probability that an arbitrary host is exploitable on the probability that an arbitrary host is compromised. This figure shows that both z and p_e are directly related to p_c, which is how we want the algorithm to behave because this means that smaller zones comprising less exploitable hosts are more secure. Also, p_e amplifies the effect of z on p_c: When $p_e = 0.1$, p_c's range is only 0.019; on the other hand, the range is 0.141 when $p_e = 0.9$, which is almost an order of magnitude increase.

Figure 5 shows the impact of the maximum zone size and the probability that an arbitrary host is exploitable on the probability that an arbitrary host can reach back to its C2 node. This figure shows that both z and p_e are directly related to p_r, which is how we want the algorithm to behave because this means that smaller zones comprising less exploitable hosts are more secure. Figure 5 shows that p_e amplifies p_r in the same fashion it does to p_c.

Figure 6 shows the impact of the maximum zone size and the interzone porosity on the probability that an arbitrary host

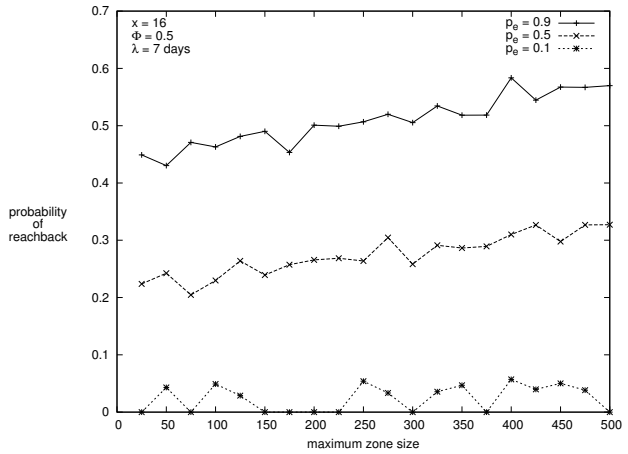

Fig. 5. Probability of reachback versus zone size and probability of exploitability.

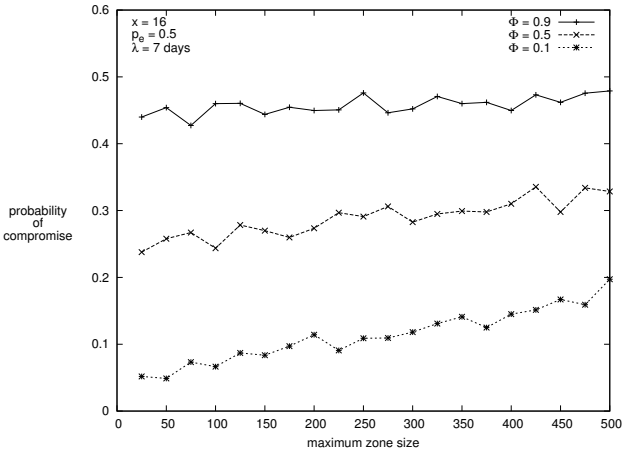

Fig. 6. Probability of compromise versus zone size and interzone porosity.

is compromised. This figure shows that both z and Φ are directly related to p_c, which is how we want the algorithm to behave because this means that smaller zones with less porous boundaries are more secure. In contrast to p_e, Φ attenuates the effect of z on p_c. When $\Phi = 0.1$, p_c's range is 0.148; on the other hand, the range is only 0.052 when $\Phi = 0.9$.

Figure 7 shows the impact of the maximum zone size and the interzone porosity on the probability that an arbitrary host can reach back to its C2 node. This figure shows that both z and Φ are directly related to p_r, which is how we want the algorithm to behave because this means that smaller zones with less porous boundaries are more secure. Figure 7 shows that Φ attenuates p_r in the same fashion it does to p_c.

Figure 8 shows the impact of the maximum zone size and the mean request interval on the probability that an arbitrary host is compromised. This figure shows that while z is directly related to p_c, λ appears unrelated. This is how we want the algorithm to behave: Smaller zones are more secure, but the rate of new collaborations should not impact security.

Figure 9 shows the impact of the maximum zone size and

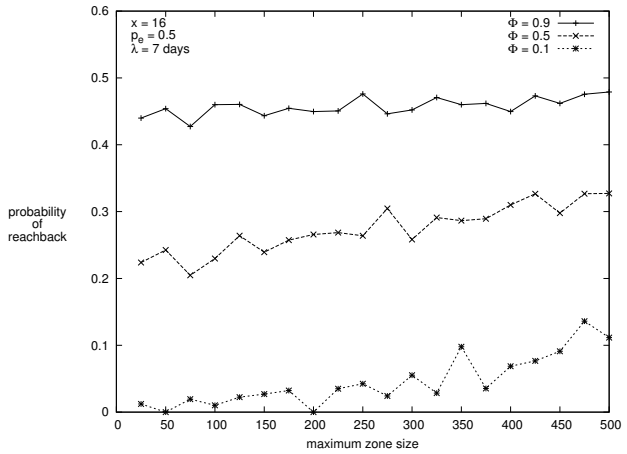

Fig. 7. Probability of reachback versus zone size and interzone porosity.

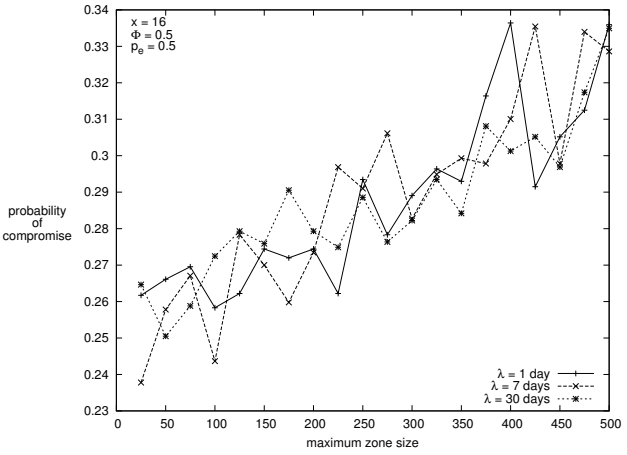

Fig. 8. Probability of compromise versus zone size and mean request interval.

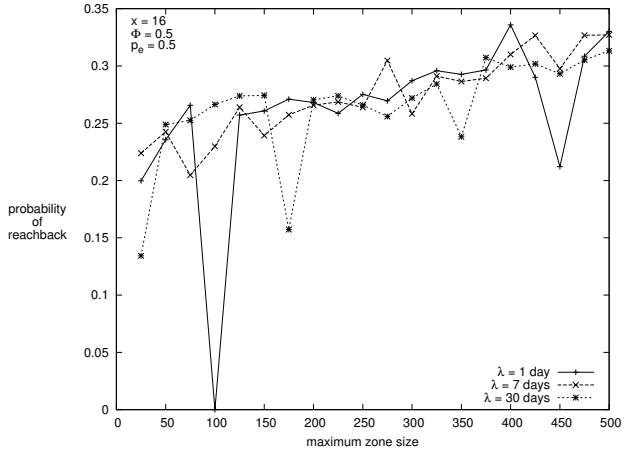

Fig. 9. Probability of reachback versus zone size and mean request interval.

the mean request interval on the probability that an arbitrary host can reach back to its C2 node. This figure shows that while z is directly related to p_r, λ appears unrelated. This is

how we want the algorithm to behave: Smaller zones are more secure, but the rate of new collaborations should not impact security.

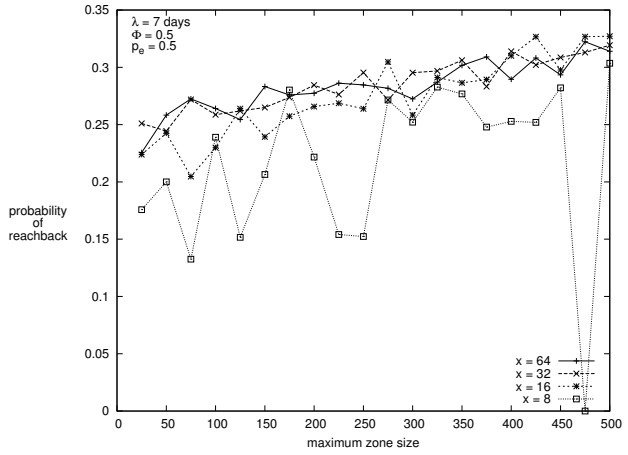

Fig. 10. Probability of reachback versus zone size and externally-facing host count.

Figure 10 shows the impact of the maximum zone size and the externally-facing host count on the probability that an arbitrary host can reach back to its C2 node. This figure shows that both z and x are directly related to p_r, which is how we want the algorithm to behave because this means that smaller zones with fewer Internet gateways are more secure. Also, Figure 10 shows that while p_r is sensitive to x, its effect diminishes as x grows. While the gap between the curves for $x = 8$ and $x = 16$ is clear, the gap between the curves for $x = 32$ and $x = 32$ is less distinct.

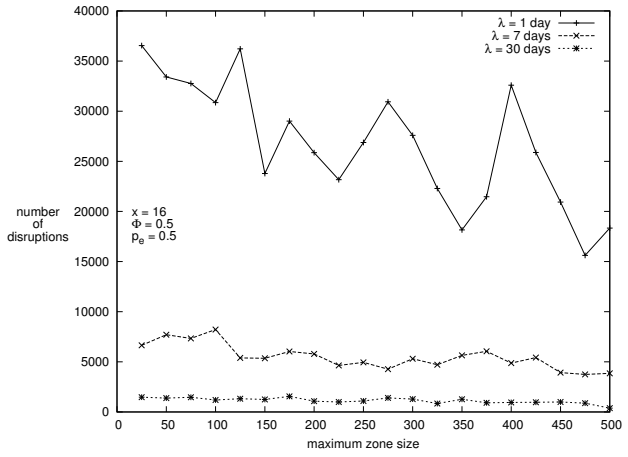

Fig. 11. Disruptions versus zone size and mean request interval.

Figure 11 shows the impact of the maximum zone size and the mean request interval on the number of disruptions. This figure shows that both z and λ are inversely related to distraction count, which is how we want the algorithm to behave because this means that larger zones with less frequent new collaborations are less disruptive for the defender. Also, λ attenuates the effect of z on d. When $\lambda = 1$ d, d's range is 20932; on the other hand, the range is only 1175 when $\lambda = 30$ d.

VI. Conclusions

In this paper, we make three basic contributions. First, we propose an MTD intrusion tolerance technique based on dynamically assigning hosts in a network to cyber zones and restricting the traffic flow between these zones. We also provide a reference implementation for this technique. Finally, we study how dynamic cyber zone defense impacts the attacker and defender using a discrete-time simulation.

There is a substantial list of enhancements we look forward to making to this work. First, we will consider a ZM that batches requests and finds zone assignments that are minimally disruptive. We will also collect real data regarding patterns of human and host collaboration to replay in our simulation. Third, we will relax assumptions that we make in the discrete-time simulation: we will allow the ZM to create new zones, we will allow many-to-many relationships between zones and charge numbers and we will allow many-to-many relationships between humans and hosts.

References

[1] https://pypi.python.org/pypi/simpy.

[2] Marci McBride and Robert Mitchell. A Zoning Algorithm for Dynamic Cyber Zone Defense. In *Computing and Communication Workshop and Conference*, Las Vegas, Nevada, USA, January 2017.

[3] Robert Mitchell. Epidemic-Resistant Configurations for Intrusion Detection Systems. In *Communications and Network Security*, Las Vegas, Nevada, USA, October 2017.

[4] Robert Mitchell and Paul Sery. Refining the Foundations for Cyber Zone Defense. In *New Technologies, Mobility and Security*, Larnaca, Cyprus, November 2016.

[5] Robert Mitchell, Paul Sery, and Tom Klitsner. Foundations for Cyber Zone Defense. In *International Conference on Computer Communication and Networks*, Waikoloa, Hawaii, USA, August 2016.

[6] Robert Mitchell and Elizabeth Walkup. Further Refinements to the Foundations of Cyber Zone Defense. In *MILCOM*, Baltimore, Maryland, USA, October 2017.