BGPeek-a-Boo: Active BGP-based Traceback for Amplification DDoS Attacks

Johannes Krupp
CISPA Helmholtz Center for Information Security
Saarbrücken, Saarland, Germany
johannes.krupp@cispa.saarland

Christian Rossow
CISPA Helmholtz Center for Information Security
Saarbrücken, Saarland, Germany
rossow@cispa.saarland

2021 IEEE European Symposium on Security and Privacy (EuroS&P) | 978-1-6654-1491-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/EUROSP51992.2021.00036

***Abstract*—Amplification DDoS attacks inherently rely on IP spoofing to steer attack traffic to the victim. At the same time, IP spoofing undermines prosecution, as the originating attack infrastructure remains hidden. Researchers have therefore proposed various mechanisms to trace back amplification attacks (or IP-spoofed attacks in general). However, existing traceback techniques require either the cooperation of external parties or *a priori* knowledge about the attacker.**

We propose BGPEEK-A-BOO, a BGP-based approach to trace back amplification attacks to their origin network. BGPEEK-A-BOO monitors amplification attacks with honeypots and uses *BGP Poisoning* to temporarily shut down ingress traffic from selected Autonomous Systems. By systematically probing the entire AS space, we detect systems forwarding and originating spoofed traffic. We then show how a graph-based model of BGP route propagation can reduce the search space, resulting in a $5\times$ median speed-up and over $20\times$ for $1/4$ of all cases. BGPEEK-A-BOO achieves a unique traceback result 60% of the time in a simulation-based evaluation supported by real-world experiments.

***Index Terms*—Amplification DDoS, BGP Poisoning, Traceback, IP Spoofing**

1. Introduction

Amplification attacks [1] continue to be one the most powerful type of DDoS attacks, reaching attack bandwidths as high as 1.7 Tbps in 2018 [2] or 2.3 Tbps in 2020 [3]. These attacks rely on *IP spoofing*: As IP header information is not authenticated, crucial fields such as the packet's source address can be set to arbitrary values by the attacker. Even worse, IP spoofing not only enables these attacks in the first place, it also effectively hides the attack's origin. Without knowing the true network origin, identifying the actors behind these attacks is nigh impossible. Thus, a traceback mechanism for these attacks is of prime importance.

Previous traceback approaches for amplification attacks can only link attacks to scanners used in attack preparation [4] or re-identify attacks from known sources [5]. This restricts them to the subset of incidents where rich auxiliary information is known *a priori*. Approaches to trace back IP spoofing in general [6] are mostly based around the idea of packet marking [7]–[12], where routers encode path information in the packet header, or collecting flow telemetry data [13]–[17]. However, both require the cooperation of a large number of routers along the path and thus a widespread deployment on the Internet—something we have not seen despite these approaches being known for over a decade.

In this paper we propose BGPEEK-A-BOO, a novel approach to trace back amplification attacks that requires neither cooperation of on-path routers nor knowledge of potential attack sources. The main insight behind our approach is as follows: While attackers may spoof IP level information, they are usually tightly coupled to a given spoofing-capable network location. Packets sent by the attacker are thus bound to the routes chosen by their network provider. This allows us to use the Border Gateway Protocol (BGP) to identify the Autonomous System (AS) that emits the spoofed attack traffic, which constitutes a fundamental step towards fighting these attacks: Once identified, prosecutors can contact the AS operators to investigate the perpetrators behind the attack, which must be customers of the AS. Further, the spoofing AS can be pressured into implementing egress filtering by its peers, similar to what happened to McColo in 2008 (cf. [18]).

BGPEEK-A-BOO, shown in Figure 1, consists of a number of amplification honeypots organized in multiple $/24$ prefixes and a BGP router that can advertise routes for these prefixes. The honeypots emulate services that are vulnerable to amplification in order to be selected as reflectors in amplification attacks [19]. During attacks, these honeypots will receive spoofed traffic sent by the attacker. Through *BGP Poisoning* we can then exclude certain ASes from propagating routes towards our system. In particular, depriving the attacker of a route causes the spoofed traffic to either switch to an alternative route, which may be observed by a change in TTL values at the honeypots, or to cease entirely. Building on this observation, we systematically probe ASes to uncover those involved in forwarding attack traffic—eventually leading us to the spoofing AS itself.

In a second step, we show how AS relationship data can be used to drastically limit the search space. For this we build a *BGP flow graph* that captures how BGP advertisements propagate and analyze which systems are *reachable* and *dominated* by others. We find that both algorithms achieve a perfect attribution result 100% of the time in an idealized, and still over 60% in a more realistic simulation. Our naive algorithm requires a median of 549 BGP Poisoning steps (91.5 hours) for traceback, while our graph-based algorithm improves this to 98.5 steps (16.4 hours), with 25% of cases even terminating in at most 29 steps (4.8 hours). An 8-fold parallelization of our methodology reduces the median traceback duration to less than an hour.

In summary, our contributions are the following:

1) We propose a novel approach for AS-level traceback of IP spoofing by leveraging BGP Poisoning. Our approach requires neither cooperation of external parties nor a priori knowledge about the attacker.
2) We present two traceback algorithms, showing that BGP-based traceback is feasible in principle and can be greatly sped up when augmented with AS relationship data.
3) We provide an extensive simulation-based evaluation, measuring the influence of various parameters on performance and correctness. We confirm our simulator through real-world experiments using the PEERING BGP testbed [20] and RIPE Atlas [21].

2. Background

In this section, we give a brief recap on amplification attacks and BGP Poisoning.

2.1. Amplification DDoS Attacks

In an amplification DDoS attack, the attacker tricks public UDP services (e.g., DNS servers) into sending large amounts of traffic to the victim. This is possible, because IP spoofing enables an attacker to spoof the source address of packets to be the address of the *victim*. The service will then perceive this packet as a legitimate request and respond to the victim (making the service an involuntary *reflector*). By carefully selecting reflectors that send large responses, the attacker can maximize the traffic that is reflected to the victim and achieve traffic *amplification*. IP spoofing further hides the attacker's (network) location, which makes finding the attacker behind an amplification attack notoriously difficult.

Fortunately, amplification attacks have been identified to largely be launched from *single* sources such as Booter services [5], [19]. For attackers, finding such reliable and capable source infrastructures is challenging. Consequently, these infrastructures are usually reused over long time spans. These services have also been reported to reuse the same set of reflectors for multiple attacks over an extended period of time [5]. We can therefore assume that reflectors continuously[1] receive spoofed attack traffic from the same origin.

2.2. The Border Gateway Protocol (BGP)

The Internet is often described as a "network of networks", as it comprises thousands of so-called *autonomous systems* (ASes). Every AS is a network under the control of a single entity. An AS is usually responsible for a number of IP prefixes and can be identified by its unique *AS number* (ASN).

BGP [22] enables routing between ASes. In BGP, ASes exchange routing information with their neighbors through route advertisements (also called announcements). Each route advertisement describes a path of ASes (AS_PATH) via which a certain IP prefix may be reached. When a BGP router receives an advertisement for a prefix, it first checks if it already knows a *better*[2] route for that prefix. If it does, the new route is only kept as a fallback. Otherwise, the router prepends its own ASN to the AS_PATH and advertises this new route to its neighbors. Between two BGP routers, a new advertisement for a prefix also implicitly withdraws the old route advertised for that prefix. When routing traffic, packets are forwarded according to the best known route for the most specific prefix covering the traffic's destination. We assume that the attacker does not control an entire AS, but is a customer of an RFC-compliant AS.

1. As we will show in Section 6.3.4, even though individual attacks might be too short for BGP-based traceback, we can aggregate multiple attacks to the same origin, thereby fulfilling this demand.

2. according to its operator defined policies

2.2.1. BGP Poisoning:. BGP detects and prevents loops [22]. Before considering new advertisements, routers check if their own ASN is already included in the AS_PATH. If so, the new advertisement will be considered as a withdrawal only and no longer propagated to the AS's neighbors. A side-effect of loop detection is that it enables *BGP Poisoning*. By crafting the AS_PATH to include other systems' ASNs, loop detection can intentionally be triggered at these other systems. Specifically, to trigger loop detection at ASes $X_1, \ldots, X_n$, an AS A may send an advertisement with

$$\texttt{AS\_PATH} = (A, X_1, \ldots, X_n, A)$$

If any AS $\in X_1, \ldots, X_n$ receives this advertisement, it will find itself already present in the AS_PATH, consider this a "loop", and handle it as a withdrawal subsequently. The first A ensures that A's neighbors correctly see A as the next on-path AS, while the last A ensures that the prefix is still seen as originating from A (as required, e.g., for Route Origin Validation [23], [24]). We will call A the *poisoning AS* and X_1 through X_n the *poisoned ASes*. Despite its negative name, BGP Poisoning does not imply malevolence—after all, dropping advertisements can only impede reachability of the *poisoning* AS, but not others. On the contrary, since it gives operators a way to control *in*bound traffic paths, its utility has been proven for many traffic engineering tasks [25]–[27].

3. BGP-based Traceback

As noted in Section 2.2.1, BGP Poisoning can be used to discard route advertisements at other ASes. In this section we show how the resulting side-effects can be leveraged to find the origin AS of spoofed attack traffic and present our traceback system BGPEEK-A-BOO.

3.1. Poisoning for Traceback

Assume an AS A is sending spoofed traffic to a reflector and receives a poisoning advertisement for the reflector's prefix. Since the AS will handle this advertisement like a withdrawal, it will remove its routing information for that prefix. However, without routing information it can no longer send traffic to the reflector. The reflector will hence stop receiving traffic from A.

A similar observation can be made for ASes *forwarding* traffic to the reflector. Assume an AS F is normally

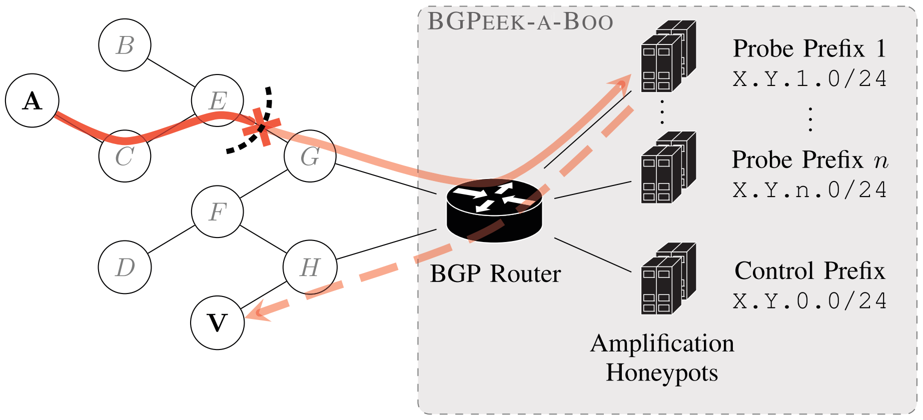


Figure 1: BGPEEK-A-BOO overview. After poisoning E, the attack towards V is no longer observed at the amplification honeypots and must therefore originate from either A, B, C, or E.

forwarding spoofed traffic from A to the reflector and receives a poisoning advertisement. Next to losing the ability to send traffic to the reflector, F also has to withdraw any routes for that prefix it had advertised to its neighbors. Thus A can no longer route traffic to the reflector via F. If no alternative path from A to the reflector exists that circumvents F, then A again loses connectivity and traffic at the reflector will stop. However, even if an alternative path exists, this case might be observable at the reflector: Unless the IP hop count along both paths is exactly the same, the traffic's TTL value at the reflector will change.

This allows us to check whether some AS X was *on-path* of a traffic flow: If poisoning X causes the traffic to stop or its TTL value to change, X *was* on-path. Furthermore, if traffic stopped, the origin AS has no alternative routes avoiding the poisoned AS X.

3.1.1. Default Routes. In practice, some ASes can still route traffic even for prefixes that they have no explicit routing information for. These *default routes* can be either configured statically or an upstream AS can advertise itself as the next hop for a large prefix (e.g., `0.0.0.0/0`). In these cases, the poisoned AS will not loose connectivity entirely, but switch to its default route. As before, such a route change can result in an observable TTL change. Furthermore, since a default route cannot be advertised with a more specific prefix than the original route, BGP will avoid paths including a default route whenever possible.

3.1.2. Combined Probes. The on-path check can also be performed for multiple ASes simultaneously using a combined poisoning advertisement. If the poisoning of an AS leads to a stop in traffic, poisoning additional ASes cannot undo this effect. If, however, it leads to a TTL change only, poisoning additional ASes may cause further re-routing or even eliminate alternative paths. While this could negate the TTL change under specific circumstances[3], it results in additional traffic stops in many cases.

Ultimately though, a combined probe can only tell whether *any* of the poisoned ASes were on-path. To find the exact on-path AS within the probed set, we can use a binary search approach, iteratively splitting the probing set in half and repeating the probing for each half.

3. if it causes traffic to take a path with the exact same IP hop count as the original path

3.1.3. Active Measurements. This technique can be further supplemented by active measurements. By provoking replies from a host in the measured AS (e.g. through ICMP Pings or TCP SYNs[4]), we can observe the effect of poisoning on the AS. If the replies to our active measurements stop under poisoning, but the spoofed traffic continues (or vice versa), we can conclude that the measured AS was *not* the spoofing source.

3.1.4. Probes in BGPEEK-A-BOO. BGPEEK-A-BOO uses all but one of its prefixes as *probe prefixes* to probe network responses to poisoning advertisements. The remaining prefix is designated as the *control* prefix and will only be advertised regularly (non-poisoned). We will refer to honeypots in probe prefixes and the control prefix as *probe honeypots* and *control honeypots* respectively.

To probe an AS, the BGP router of BGPEEK-A-BOO sends a poisoned route advertisements for a probe prefix that receives attack traffic. Once the routes have stabilized, it records the impact on the attack traffic and on pings. If the attack is also observed by some control honeypots, impact can also be measured by comparing traffic between the probe and control honeypots.

4. A naive Traceback Approach

Using these measures we propose the naive traceback algorithm depicted in Figure 2. The algorithm simply loops over all ASes (set $\mathcal{A}$) in chunks of size at most n and considers each chunk as a combined probe $\mathcal{P}$.

The actual probing is performed by PROBE, which sends out a poisoned advertisement for the ASes in $\mathcal{P}$, performs the active measurements, and returns the observed effects. It returns the effect on the attack traffic (r_{passive}) and the active measurement results ($\vec{r}_{\text{active}}$). r_{passive} can be either `NO_EFFECT` if no change was observed, `TTL_CHANGE` if we observed a TTL change, or `STOP` if traffic has stopped entirely. Similarly, $\vec{r}_{\text{active}}$ is a

4. suitable candidates and ports could be found through Internet-wide scanning or by leveraging a search engine such as Shodan [28]

```
procedure NaiveTraceback($\mathcal{A}, A, n$)
    $\mathcal{C} \leftarrow \emptyset$                                        ▷ candidates
    for block $\mathcal{P}$ in $\mathcal{A}$ of size $\leq n$ do
        ProbeAndUpdate($\mathcal{P}$)
    return $\mathcal{C}$

procedure ProbeAndUpdate($\mathcal{P}$)
    $r_{\text{passive}}, \vec{r}_{\text{active}} \leftarrow$ Probe($\mathcal{P}$)
    if $r_{\text{passive}} \neq$ NO_EFFECT then
        $\mathcal{P} \leftarrow$ Update($\mathcal{P}, r_{\text{passive}}, \vec{r}_{\text{active}}$)
        if $|\mathcal{P}| = 1$ then
            $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{P}$                  ▷ AS in $\mathcal{P}$ was on-path
        else if $|\mathcal{P}| \geq 2$ then
            $\mathcal{P}_1, \mathcal{P}_2 \leftarrow$ Split($\mathcal{P}$)           ▷ “Binary Search”
            ProbeAndUpdate($\mathcal{P}_1$)
            ProbeAndUpdate($\mathcal{P}_2$)

procedure Update($\mathcal{P}, r_{\text{passive}}, \vec{r}_{\text{active}}$)
    if $r_{\text{passive}} =$ STOP then
        $\mathcal{P}_{\text{inconsistent}} \leftarrow \{X \in \mathcal{P} \mid \vec{r}_{\text{active}}[X] \neq \text{STOP}\}$
    else
        $\mathcal{P}_{\text{inconsistent}} \leftarrow \{X \in \mathcal{P} \mid \vec{r}_{\text{active}}[X] = \text{STOP}\}$
    return $\mathcal{P} \setminus \mathcal{P}_{\text{inconsistent}}$
```

Figure 2: Naive traceback algorithm

vector with one component (either NO_EFFECT or STOP) per probed AS.

If poisoning of $\mathcal{P}$ shows an effect on the attack traffic, the probe is then narrowed down to find the exact AS(es) that caused this effect. For this, first, all probed ASes that show an inconsistent behavior in their *active* measurements are discarded from the probe (Update). If this already reduces the probe to a single consistent AS, it is then added to the candidate set $\mathcal{C}$ as a confirmed on-path AS. Otherwise, the probe is split in half (Split) and the probing is repeated for each half recursively. Once all ASes have been probed that way, the final set of confirmed on-path ASes $\mathcal{C}$ is returned.

4.1. Runtime Analysis

The runtime of this traceback approach is greatly dominated by the probing step, which involves sending out a poisoned advertisement and performing active measurements. This is because (1) it may take several minutes for routes to settle after a new advertisement [26], [29], only after which active measurements can be performed, and (2) several BGP mechanisms further limit the rate at which routers may send out new advertisements (Section 7.1). Therefore, realistically, advertisements cannot be made much faster than once every ten minutes.

We will thus count the number of required probing steps to analyze the traceback runtime. At chunk size n and a total of N ASNs in $\mathcal{A}$, the naive traceback algorithm from Figure 2 takes $\lceil \frac{N}{n} \rceil$ steps to test each chunk once, plus an additional $2 \log_2 n$ steps for every on-path AS to reduce the chunk it is contained in down to a single AS. For example, the AS65000 BGP Routing Table Analysis Report [30] lists roughly 66000 active ASes for the end of 2019 and an average AS path length of 5.5. With a chunk size of $n = 128$ this thus gives an average of 593 steps total, which, at 6 advertisements per hour, would take about 4 days and 3 hours to complete.

Given that spoofing sources, such as Booter services, are active for extended periods of time (see Section 6.3.4) and reuse the same amplifiers for a week or longer [5], this shows that BGP-based traceback is feasible in principle.

4.2. Discussion

While this algorithm is intuitive and requires no external knowledge of AS properties or relationships, it comes with two main drawbacks: (1) It only returns an unordered set of on-path ASes, which still leaves the exact path and origin unknown. (2) It “wastes” a lot of time poisoning off-path ASes that could potentially be avoided, as it effectively conducts an exhaustive search over the entire AS space.

4.2.1. On-Path Ordering. Ideally, one would like to find the true origin AS of the spoofed traffic, or at least the on-path AS closest to it that can still be discovered. However, through probing we can only tell *whether* an AS was on-path or not, but not its position along the path.

While at first glance it seems that this problem could be solved by comparing TTL values received from hosts located in these ASes, this is not necessarily true: Although the AS level path should be the same for both traffic originating from and traffic forwarded by an AS, the IP level paths (and hence their hop counts) can differ vastly. In a similar vein, one might attempt to infer the on-path order from traceroutes towards hosts in the candidate ASes, mapping the obtained IP level traceroute paths into AS level paths. Barring the problems of mapping IP to AS level paths [31], [32], traceroutes from the vantage point can only reveal paths *towards* other hosts, but not their reverse paths, which we are interested in.

4.2.2. Runtime Improvement. Although the estimated runtime does not seem prohibitive, the question still remains whether additional knowledge about ASes, such as their relationships, can be used to achieve effective traceback more efficiently. For example, as a simple optimization the search can be aborted as soon as a stub AS is confirmed to be on-path. Since stub ASes do not provide transit for other ASes, an on-path stub AS must be the one originating the observed traffic. In these cases, such an early termination will reduce the expected runtime to $1/2$ of the original algorithm. In the next section, we show how the runtime can be further improved by leveraging AS relationship data as well as the information about alternative path availability one can obtain from probing.

5. Flow Graph based Traceback

Our graph-based traceback algorithm exploits knowledge about the relationship between ASes to limit the search space. For example, if the attack traffic stops under poisoning, we know that the source has no alternative route that avoids the poisoned ASes. To efficiently reason about alternative paths and whether an advertisement might be propagated from one AS to another, we define a so-called *AS Flow Graph*.

5.1. AS Flow Graphs

AS relationships are usually classified as either *customer-provider* (*CP*) or *peer-to-peer* (*P2P*) relations [33]. In a *customer-provider* relation, one AS (the customer) pays another (the provider) for transit such that the customer may reach and be reached from the Internet via the provider. In a *peer-to-peer* relation, two ASes agree to transit traffic for their customers to one another, thereby reducing the amount of traffic they would have to pay their provider for otherwise.

The resulting BGP paths are generally assumed to be valley-free [33]: zero or more *customer-provider* edges ("up"), followed by at most one *peer-to-peer* edge ("sideways"), followed by zero or more *provider-customer* edges ("down"). In other words: a *peer-to-peer* or *provider-customer* edge can never be followed by a *customer-provider* or *peer-to-peer* edge, as this would result in a "valley". Note that this property is symmetric and holds for both, the AS-level paths taken by routed traffic as well as the propagation paths of BGP advertisements.

Following this, an AS may receive an advertisement in two states: If the advertisement was received from a customer (i.e., via a *customer-provider* edge), the valley-free assumption does not restrict the edge types that may follow. We will call this state *unconstrained*. If an advertisement was received from either a peer or a provider, it may only be forwarded to customers, but not to other peers or providers. We will thus call this state *constrained*.

We can use a graph to model advertisement propagation that uses *two* nodes per AS, one for each state. Formally, we define this graph $G = (V, E)$ as follows: Every AS A is represented by two vertices, u_A and c_A, representing the *unconstrained* and *constrained* state respectively,

$$V = \bigcup_{A \in \text{AS}} \{u_A, c_A\}$$

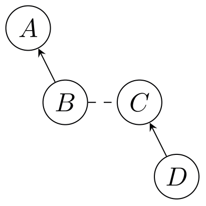
We will denote the AS represented by a node x using $\text{asn}(x)$,

$$\text{asn}(x) = A \Leftrightarrow x \in \{u_A, c_A\}$$

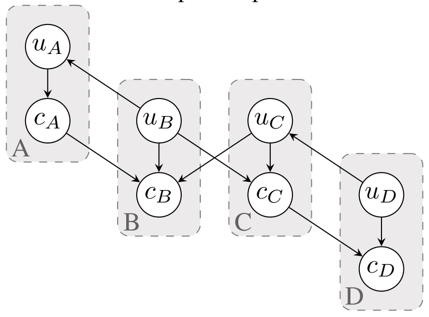
When AS A is a provider of AS B, advertisements may only be forwarded "downhill" from c_A to c_B or "uphill" from u_B to u_A. If A and B share a *P2P* relation, then advertisements may only be forwarded between A and B at the "peak" of the path, i.e., from u_A to c_B or from u_B to c_A. Finally, advertisements received at u_A may of course also be forwarded to c_A. In total,

$$E = \left(\bigcup_{A,B \in \text{CP}} \{(u_A, u_B), (c_B, c_A)\} \right) \cup \left(\bigcup_{A,B \in \text{P2P}} \{(u_A, c_B), (u_B, c_A)\} \right) \cup \left(\bigcup_{A \in \text{AS}} \{(u_A, c_A)\} \right)$$

This graph then captures all valley-free AS paths, and every path in this graph corresponds to a valid valley-free AS path. For a formal proof, please refer to Section A.



(a) Example AS relationships: A is provider of B, C is provider of D, and B and C have a peer-to-peer relation.



(b) The resulting AS flowgraph shows that route advertisements from D may reach C and B, but can never reach A.

Figure 3: AS flowgraph example

By construction, the orientation of edges denotes the direction of advertisement propagation. For example, an edge from u_B to u_A implies that advertisements received by B may be further propagated to A. However, the inverse graph obtained by reversing all edges is meaningful as well, as it describes all possible traffic flows between ASes: If AS B advertises a route for a prefix to AS A, then A may send traffic destined towards that prefix to B.

In the example given in Figure 3, A is a provider of B, C is a provider of D, and B and C have a peer-to-peer relation. However, it is not obvious from the relationship graph, whether advertisements from D may reach A. The resulting flowgraph answers this unequivocally: Advertisements from D can reach C ($u_D \rightarrow u_C$) and B ($u_D \rightarrow u_C \rightarrow c_B$), but cannot reach A, since there is no directed path from either u_D or c_D to u_A or c_A.

5.1.1. Reachability and Dominance. A poisoned advertisement can affect ASes in two ways: ASes included in the advertisements' `AS_PATH` are affected directly, as they will discard the advertisement due to loop detection. However, this also prevents them from propagating the advertisement further, which, in turn, can affect other ASes. While some of these indirectly affected ASes may still receive the advertisement via alternative routes and thus only experience a route change, others may no longer be able to receive the advertisement at all. To reason about the propagation of (poisoned) advertisements originating from a specific AS A, we can define its AS-specific subgraph:

Definition 1 (AS-specific subgraph)**.** *For the flow graph G and AS A, we define the AS-specific rooted subgraph $G_A = (V_A, E_A, u_A)$ as the subgraph of G rooted at u_A.*

We can then define two relations on this graph, reachability and (joint) dominance, to capture which ASes *might* potentially be affected by a poisoned advertisement and which *will* be inevitably.

Consider an advertisement that is poisoning ASes $P = \{X_1, \ldots, X_n\}$ (with corresponding nodes $p = \{u_{X_1}, c_{X_1}, \ldots, u_{X_n}, c_{X_n}\}$). Another AS Y might only be affected by this advertisement if it can receive advertisements from A via any AS in P[5]. As all paths in the graph G_A describe valid propagation paths for advertisements originating from A, AS Y therefore might be affected if there is a path from any node $x \in p$ to any node $y \in \{u_Y, c_Y\}$.

Definition 2 (Reachability). *We call a node y* reachable *from another node x, iff there exists a path in G_A from x to y.*

$$\begin{aligned} & y \in \mathrm{reachable}_{G_A}(x) \Leftrightarrow \\ & \quad \exists \pi = (x_1 = x, \ldots, x_n = y) : \\ & \quad\quad \forall 1 \leq i < n : (x_i, x_{i+1}) \in E_A \end{aligned}$$

The set $\mathrm{reachable}_{G_A}(x)$ describes all nodes whose traffic towards A *may* be routed via x. The definition of reachable can be trivially extended to sets of nodes by taking their union:

$$\mathrm{reachable}_{G_A}(\{x_1, \ldots, x_n\}) = \bigcup_{i=1}^{n} \mathrm{reachable}_{G_A}(x_i)$$

Therefore, the set of ASes that *might* be affected by an advertisement poisoning nodes p is simply $\mathrm{reachable}_{G_A}(p)$. That is, if poisoning of nodes p causes the TTL values of the attack traffic to change we can infer that the origin was affected and must be an element of $\mathrm{reachable}_{G_A}(p)$.

In a similar vein we can also define nodes that *must* be affected. Intuitively, a node y must be affected if it cannot receive advertisements from A but via nodes in p. Thus, once all nodes in p are removed from the graph, y should be no longer reachable from u_A. In graph theory terms p then constitutes a *u_A-y-vertex-separator* (albeit defined over directed graphs), but can also be seen as a generalization of the concept of *dominators* from control-flow-graph analysis.

Definition 3 ((Joint) Domination). *We call a node y (jointly)* dominated *by a set of nodes $\{x_1, \ldots, x_n\}$ in graph G_A, iff y is reachable from u_A but removing the set $\{x_1, \ldots, x_n\}$ breaks reachability from u_A to y. Formally*

$$\begin{aligned} & y \in \mathrm{dominatees}_{G_A}(\{x_1, \ldots, x_n\}) \Leftrightarrow \\ & \quad y \in \mathrm{reachable}_{G_A}(u_A) \wedge y \notin \mathrm{reachable}_{G'_A}(u_A) \end{aligned}$$

where $G'_A = (V'_A, E'_A, u_A)$ with $V'_A = V_A \setminus \{x_1, \ldots, x_n\}$ and $E'_A = \{(x, y) \in E_A \mid x \in V'_A \wedge y \in V'_A\}$.

Hence, the set of ASes that *will* inevitably be affected, i.e., those that will have to switch to default routes or possibly loose connection, by an advertisement poisoning nodes p is $\mathrm{dominatees}_{G_A}(p)$. As noted above, other ASes from $\mathrm{reachable}_{G_A}(p)$ might be affected as well, but for those contained in $\mathrm{dominatees}_{G_A}(p)$ we can be certain.

5. Under specific circumstances, poisoned advertisements may also induce route changes at ASes that have no connection to a poisoned ASes. We analyze these cases in Section 7.4.

5.2. Flow Graph based Traceback

We can leverage this flow graph for AS traceback in two ways: First, given a known on-path AS we know that the neighboring on-path AS must also be one of its successors in the graph. Instead of globally searching for the origin AS we can therefore iteratively find all on-path ASes one-by-one by probing the successors of the latest found on-path AS.

Second, we can use reachability and dominance relations given by the graph to reduce our probing search space: If traffic stops, we know that the origin AS has no alternative paths available and can thus limit our search space to nodes dominated by the probed ASes (which includes the probed ASes themselves). If we observe a TTL change, we can still infer that the origin AS is at least indirectly affected by the probe and can therefore limit our search space to those nodes that are reachable from the probed ASes. We can also use similar inferences on the results of our active measurements: When responses to active measurements for a probed AS stop but the attack traffic does not, we can exclude all nodes dominated by the probed AS from our search space, as those would no longer be able to send traffic as well. Vice versa, if the attack traffic stops but active measurements show that a probed AS is still replying to pings, we can exclude all nodes reachable by the probed AS, as they also could send traffic to us via the probed AS.

5.2.1. Graph based Traceback Algorithm. Combining both of these effects leads to the graph-based traceback algorithm shown in Figure 4. Given the rooted subgraph $G_A = (V_A, E_A, u_a)$ of the receiving AS A and a probe size limit n, it returns a set of candidate source ASes.

For this, the algorithm maintains a set of source candidates $\mathcal{C}$, which is initially set to all ASes reachable from A and then continuously narrowed down during traceback. Further, it also keeps a logbook $\mathcal{L}$ of ASes that have already been probed and can thus be excluded from further probing, as well as the most recent on-path AS L, which is initially set to A. As long as there are candidates that have not been probed yet ($\mathcal{C} \setminus \mathcal{L}$), a new set of ASes to probe is selected (see Section 5.2.2) and passed to PROBEANDUPDATE, which will perform the probing and update the candidate set and logbook accordingly.

As before, probing is performed by PROBE (see Figure 2). Updating the candidate set is performed in two steps: First, UPDATEPASSIVE reduces the candidate set based on the overall effect on the attack traffic, limiting $\mathcal{C}$ to the set of nodes dominated or reachable by the current probe if traffic stopped or a TTL change was observed. Second, UPDATEACTIVE further narrows down the candidate set and the current probe by finding probed ASes whose active measurements are inconsistent with the overall observation and then removing nodes dominated or reachable by these.

If there has been an effect on the attack traffic and multiple probed ASes exhibit a consistent behavior, they are split in half (SPLIT) and the probing is repeated for each half. If a probe has been narrowed down to a single consistent AS, this AS is set as the most recent on-path AS L and a new probe is selected.

```
procedure TRACEBACK($G_A, n$)
    $\mathcal{C} \leftarrow \{\text{asn}(v) \mid v \in V_A\}$        ▷ candidates
    $\mathcal{L} \leftarrow \emptyset$        ▷ logbook
    $L \leftarrow A$        ▷ most recent on-path AS
    while $\mathcal{C} \setminus \mathcal{L} \neq \emptyset$ do
        $\mathcal{P} \leftarrow$ PICKPROBE($\mathcal{C}, \mathcal{L}, L, n$)
        $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{P}$
        PROBEANDUPDATE($\mathcal{P}$)
    return $\mathcal{C}$

procedure PROBEANDUPDATE($\mathcal{P}$)
    $r_{\text{passive}}, \vec{r}_{\text{active}} \leftarrow$ PROBE($\mathcal{P}$)
    UPDATEPASSIVE($\mathcal{P}, r_{\text{passive}}$)
    UPDATEACTIVE($\mathcal{P}, r_{\text{passive}}, \vec{r}_{\text{active}}$)
    if $r_{\text{passive}} \neq$ NO_EFFECT then
        if $|\mathcal{P}| = 1$ then
            $L \leftarrow X \in \mathcal{P}$        ▷ AS $X$ was on-path
        else if $|\mathcal{P}| \geq 2$ then
            $\mathcal{P}_1, \mathcal{P}_2 \leftarrow$ SPLIT($\mathcal{P}$)        ▷ “Binary Search”
            PROBEANDUPDATE($\mathcal{P}_1$)
            PROBEANDUPDATE($\mathcal{P}_2$)

procedure UPDATEPASSIVE($\mathcal{P}, r_{\text{passive}}$)
    if $r_{\text{passive}} =$ STOP then
        $\mathcal{C} \leftarrow \mathcal{C} \cap \text{dominatees}_{G_A}(\mathcal{P})$
    else if $r_{\text{passive}} =$ TTL_CHANGE then
        $\mathcal{C} \leftarrow \mathcal{C} \cap \text{reachable}_{G_A}(\mathcal{P})$

procedure UPDATEACTIVE($\mathcal{P}, r_{\text{passive}}, \vec{r}_{\text{active}}$)
    if $r_{\text{passive}} =$ STOP then
        $\mathcal{P}_{\text{inconsistent}} \leftarrow \{X \in \mathcal{P} \mid \vec{r}_{\text{active}}[X] \neq \text{STOP}\}$
        $\mathcal{C} \leftarrow \mathcal{C} \setminus \text{reachable}\, G_A(\mathcal{P}_{\text{inconsistent}})$
    else
        $\mathcal{P}_{\text{inconsistent}} \leftarrow \{X \in \mathcal{P} \mid \vec{r}_{\text{active}}[X] = \text{STOP}\}$
        $\mathcal{C} \leftarrow \mathcal{C} \setminus \text{dominatees}_{G_A}(\mathcal{P}_{\text{inconsistent}})$
    $\mathcal{P} \leftarrow \mathcal{P} \setminus \mathcal{P}_{\text{inconsistent}}$
```

Figure 4: Flow Graph based traceback algorithm

5.2.2. Probe Selection. Probe Selection (PICKPROBE) picks a new probe based on the current candidates $\mathcal{C}$, the ASes that have been probed before $\mathcal{L}$, the last discovered on-path AS L, and the maximum probe size n. As discussed above, the next on-path AS must be a successor of L. We can thus partition our remaining search space of unprobed candidates $\mathcal{C} \setminus \mathcal{L}$ into “layers” according to their distance to the most recent on-path AS L. The next on-path AS should then be part of the nearest layer. This ordering also ensures that, should we be unable to find the direct on-path successor of L, e.g., because it exhibits no observable poisoning reaction, we only gradually expand our search scope to later on-path ASes.

Intuitively, to reduce the candidate set as fast as possible, we would like to first probe ASes that have a high impact on the candidate set. While a stub AS could also have a high impact if it is on-path (effectively reducing the candidate set to a single entry), statistically speaking, in most cases probed ASes will be off-path ASes. The majority of candidate set reductions will thus occur in UPDATEACTIVE. We therefore rank ASes by the number of candidates that are *reachable* through them, i.e., $|\text{reachable}_{G_A}(X) \cap \mathcal{C}|$, and then select the n highest ranking ASes as the next probe.

6. Evaluation

We next turn to evaluate the runtime and success rate of BGPEEK-A-BOO and compare both proposed algorithms. For this, we were fortunate enough to obtain a temporary ASN and a temporary /22 prefix allocations for research purposes from our regional Internet registry and were granted access to the PEERING BGP testbed [20]. However, PEERING's path length limit unfortunately proved prohibitive for any actual traceback runs of BGPEEK-A-BOO on the Internet: Although a recent study showed that advertisements even with long paths of up to 255 hops are propagated to the vast majority of the Internet [34], we were only able to send advertisements with up to 5 hops via PEERING. Since the first and last of these also had to be set to our own temporary ASN, this left us with an effective probe size of 3. Unfortunately, with this even a single run of our baseline approach would have taken over 157 days to complete. We therefore resort to simulation for a comparative evaluation of our proposed traceback approaches and use the PEERING testbed for supporting experiments.

6.1. Simulation Methodology

As noted by previous works [27], [35], [36], a complete and fully accurate model of the entire Internet cannot be obtained, as it would require exact knowledge of peering agreements and router configurations, both of which are usually regarded as trade-secrets and thus generally non-public. Simulation can therefore only be performed over approximate topology data, such as the one regularly published by CAIDA [37], and by making assumptions on router configurations.

Although the simulator by Smith et al. [27] is thankfully publicly available, we found it unsuitable for our use-case as it does model neither default routes nor TTL values along paths. Consequently, we designed our own lightweight simulator.

6.1.1. Simulator Design. Our simulator is based around the same AS flow graph described in Section 5.1. To model AS paths taken from a source to a destination, our simulator assigns every edge in the graph a random weight between 0 and 100 and then computes the shortest path. This is in line with the regular BGP decision process [22, sec. 9.1], which generally prefers shorter AS paths. The random edge weight hereby models the local preference value a network operator may set to prefer one link over another. As the graph has *two* nodes corresponding to each AS X, u_X and c_X, we ensure that the edge (u_X, c_X) is assigned the weight 0, which also ensures that the resulting AS paths are loop free—a path containing both u_X and c_X can never be shorter than the one that uses the zero-weight (u_X, c_X) edge.

The effect of a poisoning advertisement can then be simulated by temporarily removing the poisoned AS's nodes from the graph, such that they can no longer send traffic to the destination nor forward advertisements to their customers or peers.

In contrast to previous works, our simulator also attempts to model the presence of default routes, which were identified as a major culprit for discrepancies between simulations and experimental results in other BGP-based tools [34]. For this, every AS is randomly marked as either *having* a default route or not with certain probability. When simulating a poisoning advertisement, those ASes with default routes are not removed from the graph, but instead increase the weights of their incoming edges by 10000. This ensures they are no longer selected as shortest paths, unless no alternative is available—as would be the effect of less-specific prefixes. ASes with default routes also have a chance of using a secondary set of incoming edge weights when being poisoned, as otherwise their default route would always be identical to their regular route.

Finally, our simulator allows mapping AS paths to IP hop counts. For this, every step along the AS path is assigned a random hop count value drawn from a negative binomial distribution, which we found to be a good fit after analyzing traceroute results from RIPE Atlas [21] (see Section 6.3.3). In the real Internet, the IP-level path length also depends on which ingress and egress routers are taken. We therefore make this random value also dependent on the preceding and succeeding AS hop.

6.2. Results

With this simulator, we conducted multiple experiments to compare our different traceback algorithms in terms of efficacy and efficiency as well as the influence of various parameters, such as the placement of the deployment location, the presence of default routes, and the choice of probe size.

As a baseline setup for our evaluation we placed the deployment location as a customer of the PEERING testbed (AS47065), which fosters comparability with our real-world experiments (Section 6.3). We set the default route probability to 40% as a conservative approximation, given that Smith et al. [34] report a default route prevalence between 26.8% and 36.7% in their experiments. Lastly, we picked 128 as a conservative probe size, since they also report successful propagation of paths of length up to 255 [34]. The flow graph used by our simulator was based on the public CAIDA AS relationship dataset for December 2019 [37], which also covered the timeframe when our real-world experiments were performed, augmented by the peering relations listed by the PEERING testbed [20].

Every experiment was repeated 1024 times, each time with a randomly chosen AS as the traffic's origin. In addition, the simulator was also given a fresh random seed for every run, such that our results are not biased due to a single lucky weight assignment or similar. We use *naive* to refer to the naive algorithm, *naive+* for the variant with early termination, and *graph* for the graph-based algorithm.

6.2.1. Traceback Success. To analyze the efficacy of our proposed traceback, we analyzed how often BGPEEK-A-BOO succeeds in finding the origin. The naive algorithm starts with an empty candidate set and selectively adds on-path ASes. Hence we consider it successful if the true origin is contained in its final candidate set. In contrast, the graph-based algorithm assumes all ASes as candidates initially, but excludes ASes during the traceback run. Therefore, to be successful, the final candidate set must also be smaller than a certain threshold (ideally of size 1).

TABLE 1: Success rate with a final candidate set of size at most x

x	≤ 1	≤ 2	≤ 3	≤ 4	≤ 5	≤ 6	≤ 7	≤ 8
naive	**10%** ±2%	**32%** ±3%	**52%** ±3%	**59%** ±3%	**60%** ±3%	**61%** ±3%	**61%** ±3%	**61%** ±3%
naive+	**28%** ±3%	**46%** ±3%	**57%** ±3%	**60%** ±3%	**61%** ±3%	**61%** ±3%	**61%** ±3%	**61%** ±3%
graph	**58%** ±3%	**62%** ±3%	**65%** ±3%	**66%** ±3%	**66%** ±3%	**67%** ±3%	**68%** ±3%	**68%** ±3%

As shown in Table 1, we find that both algorithms have similar success rates: The naive algorithm manages to identify the true origin in $61\% \pm 3\%$[6] cases, while the graph-based traceback algorithm achieves a slightly higher success rate of $68\% \pm 3\%$ with a candidate set of 8 or less. When limiting the graph-based algorithm to a single candidate, it still manages to succeed in $58\% \pm 3\%$ cases. The naive algorithm is expected to find all on-path ASes and thus has an expected candidate set size of the average path length. For the graph algorithm, we will use a candidate set size of 8 in the following.

6.2.2. Runtime. As noted before, the traceback speed of BGPEEK-A-BOO is limited by the rate at which BGP advertisements can be sent. We therefore measure the runtime of each algorithm in the number of probing steps, with a realistic step duration of 10 minutes. As can be seen from Figure 5, the naive algorithm performs similar to the number of steps derived in Section 4.1, with a mean and median runtime of 549 steps. Adding early termination on stub-ASes slightly reduces the median to 523 steps, but more importantly reduces the average to just above 400 steps, with 25% even terminating in under 268 steps. Since it is strictly superior to the naive algorithm, we will only report results for the naive+ algorithm in the following.

Overall, the graph-based algorithm performs best by far, with an average of 159 steps and a median of only 98.5 steps. Interestingly, a quarter of all cases terminate in less than 29 steps, around 4.83 hours at six advertisements per hour. Most notably, even in the worst case the graph-based algorithm still completes its search in 415 steps or less, thus making it faster than even the best run of the naive algorithm. This demonstrates that utilizing AS graph information substantially improves the efficiency, with a median runtime speed-up of 5.6×.

6.2.3. Prefix Parallelization. If BGPEEK-A-BOO observes an attack in multiple probing prefixes simultaneously, we can further speed up traceback by running multiple probes in parallel. While this does not necessarily reduce the total number of *advertisements*, it greatly reduces the number of *steps* and hence the total runtime. Figure 6 therefore compares using different numbers of probing prefixes, one (no parallelization), two, and eight. For the naive algorithm we see a linear speed-up, reducing the median runtime from 523 to 262 steps when using two

6. We denote the 95% Agresti-Coull confidence interval

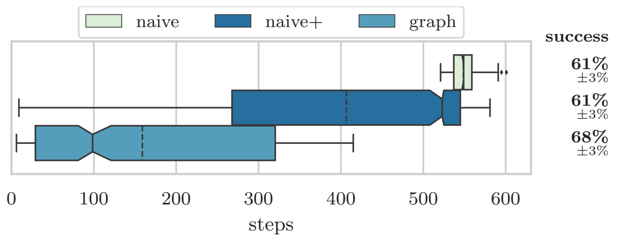


Figure 5: Runtime comparison. Dashed lines show average times, while the notch indicates the 95% confidence interval around the median.

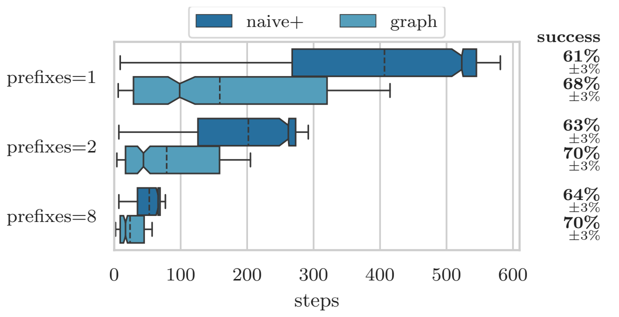


Figure 6: Influence of parallelization

prefixes and 66 steps when using eight. With the exception of the "binary search" step, probe selection is independent from previous probe results, and thus the naive algorithm can be almost perfectly parallelized. The graph-based algorithm benefits less from parallelization, since probe selection is strongly coupled to previous probe results, but still sees a 5.8× speed-up in the median runtime from 98.5 steps to 17 steps when going to eight prefixes. As with the probe size, varying the number of prefixes does not have a significant influence on the success rate.

6.2.4. Default Routes. Intuitively, default routes should have a negative influence on our traceback approach, both in terms of runtime and success. As the presence of default routes has been confirmed by multiple studies [34], [38] we would like to quantify to which extent they impact our results. To this end, we repeated the experiment two more times, once in an "ideal world" setting with no default routes and once in an exaggerated setting where 80% of ASes have a default route. In line with intuition, both algorithms generally perform better with fewer default routes, as shown in Figure 7. Interestingly though, the graph-based algorithm still terminates faster when faced with 40% default routes than the naive algorithm in the ideal world setting without any default routes. The default route prevalence is also the most determining factor of traceback success: In the idealized setting with no default routes, all simulated runs were successful, resulting in an estimated success rate of 100%±0%. Yet, even with 80% default routes, they achieve a success rate of 23% ± 3% and 29% ± 3% respectively.

6.2.5. Probe Size. We ran another experiment to measure the influence of the probe size, evaluating each algorithm with a probe size of $n = 32$, $n = 64$, and $n = 128$. The results, shown in Figure 8 confirm our intuition that the runtime of the naive algorithm scales inversely to the

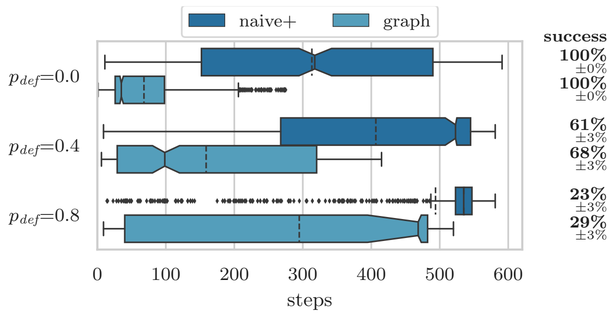


Figure 7: Influence of default route prevalence

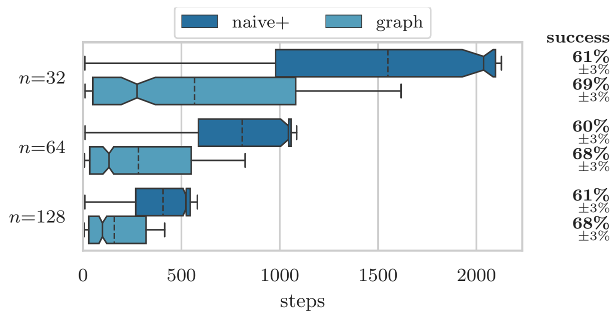


Figure 8: Influence of probe size

probe size. Where at $n = 128$ the naive algorithm has a median runtime of around 500 steps, this doubles to just over 1000 at $n = 64$ and quadruples to 2000 at $n = 32$. While the same relation holds true for the graph algorithm's *maximum* runtime, 400 steps at $n = 128$ to 1600 steps at $n = 32$, it still achieves a median runtime of less than 500 steps even with $n = 32$. At that, it outperforms both naive variants with a probe size of $n = 128$. As expected, the success rate of BGPEEK-A-BOO is not influenced by the probe size.

6.2.6. Deployment Location. To quantify the impact of the deployment location of BGPEEK-A-BOO, we also ran the experiment from three different ASes: Next to a deployment at a PEERING customer we also simulated runs from a Tier-1 provider (AS174) as well as from a national research network. However, as shown in Figure 9, neither runtime nor success rate vary significantly between the three different deployments.

6.2.7. TTL Values. BGPEEK-A-BOO relies on TTL values to detect when traffic is redirected to alternative routes. However, while mostly stable, TTL values can change for reasons unrelated to our traceback and could even be modified by an attacker attempting to evade detection. In a final experiment we therefore simulate how BGPEEK-A-BOO performs *without* TTL values, i.e., only checking whether poisoning leads to a stop in traffic. Perhaps surprisingly, the results in Figure 10 show that a lack of TTL values only leads to a slowdown of the graph algorithm, whose median runtime doubles, but has no statistically significant impact on overall traceback success. This shows that BGPEEK-A-BOO can still be used even if TTL values are found to be unreliable, albeit with increased traceback times.

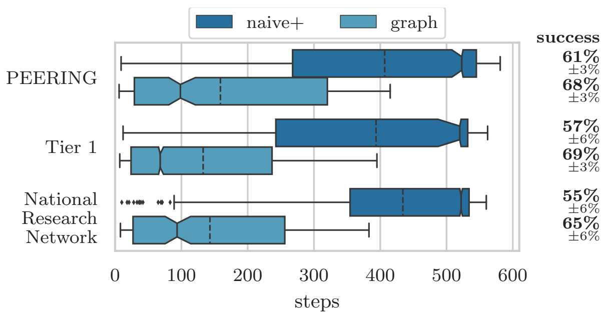


Figure 9: Influence of deployment location

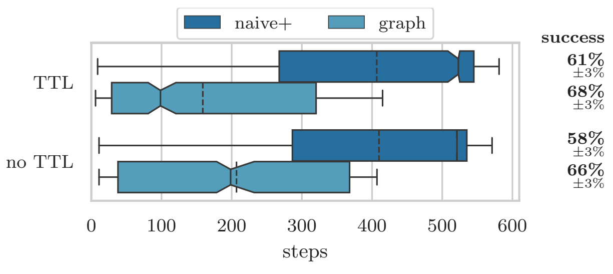


Figure 10: Influence of using TTL values

6.3. Supporting Real-World Experiments

We also leveraged the PEERING BGP testbed [20] to conduct experiments in the live Internet and analyzed attack data captured by the DDoS honeypot AmpPot [19], to bootstrap additionally required parameters and to assess the plausibility of our simulated results.

6.3.1. Changing Default Routes. To faithfully model the effect of default routes, we not only need to know how many ASes *have* a default route, but also *how often* this default route differs from the regular route. To measure this effect, we designed the following experiment: From our temporary AS we would send out a poisoning advertisement for a target system, advertising one of three $/24$ prefixes from our $/22$ allocation. The fourth prefix would be advertised regularly to serve as a control. After a short while we would then ping the target system from both, the poisoning prefix and the control prefix. If ping replies are still observed in the poisoning prefix we can conclude that the target does have a default route. If the TTL values also differ between the poisoning and the control prefix we can further infer that it differs from the regular route.

As targets we randomly selected 624 RIPE Atlas probes [21] in different ASes. TTL values were recorded five minutes after the advertisement was sent to allow routes to settle [26], [29], and new advertisements were only sent every ten minutes per prefix. Out of the 624 tested RIPE Atlas probes we found 360 (58%) to have default routes, i.e., we would still receive pings for the prefix that was advertised with a poisoning advertisement only, a fraction larger than the default route prevalence reported by Smith et al. [34] in 2020, but lower than the one reported by Bush et al. [38] in 2009. We attribute the discrepancy to two effects: First, our sample size is smaller than the one employed by both Smith et al. and Bush et al. and further biased towards ASes housing RIPE Atlas probes. Second, we found the probe's AS information as reported by RIPE's Atlas back end to not always be accurate and thus suspect that in some cases the probe's *actual* AS was different from the one poisoned, thereby making them a false positive. As noted in Section 6.2, we picked a default route probability of 40% for our simulator.

In 101 of 360 (28%) cases we further observed a discrepancy in TTL values between the poisoned and the control prefix, letting us conclude that in these cases the default route in fact differs from the regular route. At a confidence level of 95% this thus gives a probability of having a differing default route between 23% and 33%. We therefore model this in our simulator by having an AS choose a different upstream in 30% of the cases when poisoned.

6.3.2. On-Path Poisoning. As a main primitive our approach relies on the assumption that poisoning on-path ASes provokes some observable change in the target traffic, either a change in TTLs due to a route change or a complete absence of traffic due to the lack of alternative routes. To measure how this assumption holds up in practice we designed the following experiment: As the “traffic origin” we randomly selected a RIPE Atlas probe [21], and used a stream of ping packets from the probe to our traceback system deployed at PEERING as the “attack traffic”. To obtain a real-time approximation of the taken AS path, we scheduled a traceroute measurement from the Atlas probe to our traceback system. Running the traceroute in that direction ensures that we measure the actual ingress path to our system and do not have to assume paths to be symmetric. Using the Team Cymru IP to ASN Lookup [39] we then mapped traceroute IP hops to ASes and, subsequently, poisoned every discovered on-path AS one-by-one. In contrast to the previous experiment, this time we did not measure the impact on the poisoned AS, but on the ping packets from the Atlas probe simulating the target traffic flow. As before we waited five minutes before taking measurements after new advertisements and ten minutes between advertisements for the same prefix.

Note that the list of on-path ASes obtained that way is naturally incomplete, as a traceroute may miss hops along the path and the IP-to-ASN mapping leads to further inaccuracies as well [31], [32]. To validate the real-time IP-to-ASN mapping we obtained from Team Cymru we later compared the mapping results to the data covering the same timeframe published on RIPEstat [40]. Here we found 22 instances in which the IP-to-ASN mappings disagreed. Manually analyzing these 22 cases we determined the Team Cymru mapping to be correct in 14 of these, and excluded the other 8 from further analysis. As we were only interested in measuring true on-path ASes we further also excluded hops that mapped to the same AS as the Atlas probe mimicking the traffic source.

We ran the experiment with Atlas probes located in 161 different ASes, allowing us to collect a total of 327 unique (on-path AS, target AS)-pairs. In total, we found that poisoning an on-path AS resulted in a loss of traffic in 137 (42%) cases and a change in the TTL in further 112 (34%) cases. Only in 78 (24%) instances we found poisoning on-path ASes to have no measurable impact on the target traffic at all.

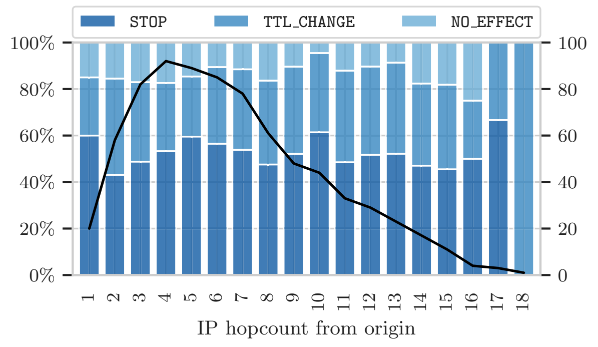


Figure 11: Real-world results of on-path AS poisoning

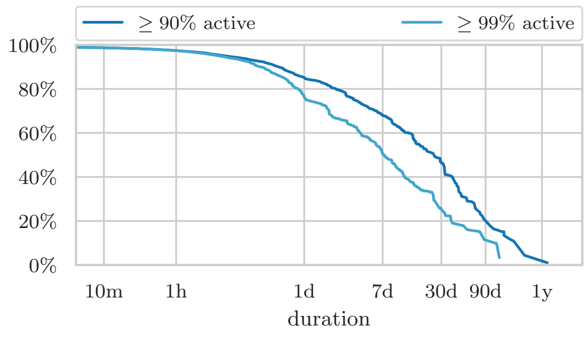


Figure 12: Fingerprint activity

To further analyze whether the impact is related on the distance between the poisoned and the measured AS, we grouped the results by their hop distance to the target AS. While ideally we would have used *AS* hop distances for this, this would have required access to the full AS path. We thus resorted to using *IP* hop distances as they could be obtained easily from the traceroute, and let the poisoning results count towards all IP hops that were mapped to the same AS. Figure 11 shows the normalized results per IP hop count, with the black line denoting the number of tested AS pairs per hop distance to give an indication of their significance. As depicted, most on-path ASes were discovered at a distance of 2 to 10 hops from the origin. However, we find that overall the distance between the poisoned AS and the traffic source appears to have little impact on how traffic is affected, and that in most cases a measurable impact can be expected.

6.3.3. TTL Distribution. We leveraged the same RIPE Atlas traceroutes to obtain a realistic model of inner-AS path lengths for our simulator, such that we can simulate IP hop counts of AS paths. For this, we utilized the same IP-to-AS mapping and then scanned the traceroute results for consecutive hops in different ASes. If we can find two of these transitions, $A \rightarrow B$ at hops $(x, x+1)$ and $B \rightarrow C$ at hops $(y, y+1)$, then the inner-AS path length of B from A to C is $y + 1 - x$ hops. We were able to extract 137 AS-triples and their corresponding hop counts and found that a negative binomial distribution with $k = 3, p = 0.62$ was a good fit.

6.3.4. Continuity of Spoofing Activity. Even with the graph-based algorithm, BGPEEK-A-BOO has an average runtime of 147 steps, or just over one day at six advertisements per hour. We therefore assess, how long an attack source may be observed consecutively. To this end, we collected data on $13,321,740$ amplification attacks observed by AMPPOT [19]. All attacks were collected between 2015-11-25 and 2020-06-15 by a *Selective Response* enabled honeypot [4]. Selective Response restricts every scanner to finding a different set of 24 of the 48 honeypot IPs, thereby imposing a unique fingerprint on the scanner. Prior work revealed a tight connection between scan and attack infrastructure [4]. We will thus use the fingerprint as an identifier for the (unknown) traffic source.

To focus on distinctive fingerprints, we only considered attacks that used at least 12 and at most 24 honeypot IPs, which left us with $8,635,257$ remaining attacks. For each fingerprint, we then computed the longest period, during which it was observed at least 90% (99%) of the time. Figure 12 shows the fraction of *attacks* whose fingerprint was observed for a given duration at a given activity level. From this plot we find that the majority of attacks stem from sources that are also active for extended periods of time. For example, over 68% of attacks stem from a source that was seen for over a week at 90% activity, 51% even at 99%.

7. Discussion

We now discuss how our evaluation results translate to the real-world deployability of our approach and the ethics of our live Internet experiments.

7.1. BGP Mechanisms

While our simulator strives to faithfully model routes and advertisement propagation, it does so by abstracting ASes and their relationships into a graph model. This abstraction may not fully model BGP in the real world, where ASes do not act as atomic entities but advertisements are instead passed between routers and processed by actual BGP implementations. As such, multiple mechanisms may influence the propagation of advertisements which we discuss below.

7.1.1. Route Flap Dampening. Route Flap Dampening (RFD) [41] aims to decrease the load on routers by maintaining a per-route penalty score. This score is increased for every update, but decays exponentially over time. Once a route's score reaches a threshold, it is no longer considered for routing nor propagated to peers until its score drops below a re-use threshold. Studies have shown that the default thresholds used in RFD can actually be harmful even for stable routes [42], and RFD was subsequently advised against [43]. Although later studies proposed new settings that have less adverse effects [44]–[46], it still remains disabled by default in, e.g., Cisco routers [47]. However, as RFD maintains a score on a *per-route* rather than a *per-prefix* basis, it does not affect our traceback technique: Whilst we send out multiple updates for the same prefix, every update includes a new AS path and therefore constitutes a new route [41, sec. 4.4.3]. Hence, even if RFD was enabled, our traceback should still work.

7.1.2. Minimum Route Advertisement Interval. Another measure to prevent high load on BGP routers is the *Minimum Route Advertisement Interval* (MRAI), that limits the rate at which updates for a certain prefix are passed on to peers. The idea behind this is that withholding routes for a certain time allows the (downstream) path exploration to converge, thereby reducing the number of updates and withdrawals sent further to peers. The recommended value for the MRAI timer is 30 seconds [22, sec. 10]. As we always waited ten minutes between advertisements this should have given routers ample time for this timer to expire. However, it also means that a real-world deployment of our approaches should employ a similar delay between advertisements.

7.1.3. Path Filtering. Smith et al. also report ASes to filter advertisements based on path lengths or by checking for potential poisoning paths [34]. For length based filters Section 6.2.5 indicates that our graph-based approach would still perform well in many cases even when limiting the path length to 32. However, filtering of poisoning advertisements can impede our traceback approach if it is performed by one of the on-path ASes. In this case, *any* poisoning advertisement may provoke the target traffic to change and thus both algorithms may falsely flag the probed AS to be on-path—even if the observed change was only caused by filtering.

7.1.4. Non-Uniform Routing. The use of active probing in our traceback assumes that all outbound packets from an AS are affected in the same way by a route change, regardless of whether they are originating from the AS or forwarded on behalf of another. In theory, every edge-router of an AS could behave differently and use a different route, which in turns means that different hosts in the AS could behave differently under poisoned advertisements. A study by Mühlbauer et al. [48] finds that such route diversity can be modelled by splitting ASes into multiple *quasi-routers*, each modelling a consistent routing behavior observed by the AS. However, they find that the vast majority in route diversity comes from prefix-dependent preferences, and that “for almost all ASes one quasi-router suffices”. Since in our case the attack traffic as well as the active measurement replies are destined towards the *same prefix*, they are not affected differently by prefix-dependent routing preferences. We can thus conclude that, for our purposes, *all* outbound traffic from an AS towards our prefix takes the same AS level path.

7.2. Observation Correlation

Our approach relies on stopping traffic and changing TTL values to infer AS-level route changes. However, TTL values along a path may also change for other reasons (e.g., inner-AS route changes), and attack traffic may cease because the attack stopped entirely. Therefore, if it is unclear whether a change was the result of a poisoning advertisement, the advertisement can be repeatedly withdrawn and re-advertised until a correlation can be confirmed or refuted. Furthermore, if BGPEEK-A-BOO’s control honeypots also observe the same attack traffic, they too can be used to decide whether a change is spurious.

7.3. AS Flow Graph Correctness

Whereas our naive algorithm makes no assumptions about AS relationships, our graph-based traceback algorithm assumes that (1) AS paths adhere to the valley-free assumption and AS relationships follow the standard customer-provider/peer-to-peer model, and (2) a global view of these relations is available. We discuss both assumptions in detail below.

7.3.1. Valley-Free Assumption and AS Relationships. While both customer-provider and peer-to-peer relations between ASes are well-established and have intuitive economic incentives, the exact relation between two ASes can be arbitrarily complex. For example, Giotsas et al. [49] identified 4026 ASes whose relationships they classified as either *hybrid* or *partial transit*. In a hybrid relation, two ASes exhibit different relations at different exchanges, whereas a partial transit relation is a restricted form of a customer-provider relation. Giotsas and Zhou [50] also find a small number of AS paths that seemingly violate the valley-free assumption, which they also attribute to non-standard AS relationships.

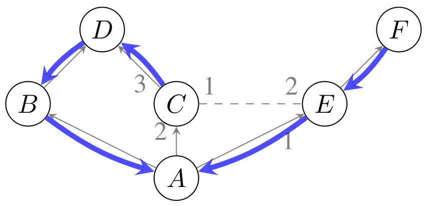
Our AS flow graph only captures customer-provider and peer-to-peer relations and requires paths to adhere to the valley-free assumption. Thus, our graph-based traceback approach may falsely exclude ASes it believes to be reachable or dominated by others when faced with such non-standard relations. However, as long as reachability and dominance information can be efficiently encoded (e.g., through adapting links in the graph), a similar algorithm may still be employed.

7.3.2. AS Relationship Dataset. AS relationships are usually subject to non-disclosure agreement and can thus only be inferred from publicly available routing data. While the state-of-the-art of inferring the global AS relationship graph has been constantly evolving [33], [51]–[58], inference will inevitably only be able to produce an approximation of the AS graph. As with non-standard AS relations, missing or incorrect links are problematic for our graph-based algorithm, which could cause it to wrongly discard ASes as candidates. In that regard, our simulation results should be seen as best-case results, as both simulator and traceback use the same graph data.

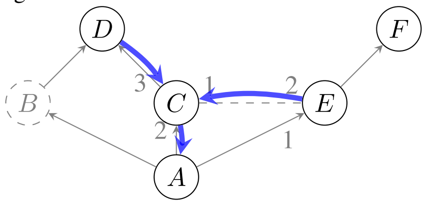
7.4. Induced Route Changes

In some cases, a poisoned advertisement can lead to route changes or losses even at ASes that can never receive an advertisement through one of the poisoned ASes. Consider for example the network shown in Figure 13. In the normal state, AS C receives three advertisements for routes to A, the direct route from A, the route $D \rightarrow B \rightarrow A$ from D, and the route $E \rightarrow A$ from E. From these it will choose the route via D, since it has the highest local preference. However, since this route is received from one of C’s providers, it cannot be exported to the peer E. AS E therefore only sees the direct route from A, which it can further advertise to its provider F.

Poisoning B makes the route $D \rightarrow B \rightarrow A$ unavailable. C therefore switches to its next preferred route, the direct route to A. Since A is a customer of C, C can



(a) A is customer of B, C, and E; B and C are customers of D; E is customer of F; D and E have a peer-to-peer relation; numbers indicate local preferences; thick arrows show the resulting routes to A.



(b) Poisoning B causes C to switch routes, which induces a route change at E and a loss of connection at F.

Figure 13: Example of induced changes

advertise this new route $C \rightarrow A$ to its peer E. This, however, induces a route change at E, because this new route has a higher local preference at E. Furthermore, since the best route at E is now received from a peer, it can no longer be exported to E's provider F. F therefore loses its connection to A. Note that neither E nor F could ever have a route to A via the poisoned AS B. Yet, they see a route change or even a loss of connectivity.

An AS can only cause such *induced* changes, if it can receive two different routes, one from a customer and one from a peer or provider. Only in that case, the set of other ASes that it can export routes to can change: Switching from a customer-provided route to a peer/provider-provided one limits it to advertise this route to its customers, switching the other way enables it to also advertise a route to its peers and providers. We will call these ASes *ambiguous*. ASes that are reachable through an ambiguous AS may therefore experience induced changes. Further, if an AS is *only* reachable through peers or providers of ambiguous ASes, it may also experience an induced connectivity loss.

For our naive algorithm, such induced changes can cause additional ASes to appear in the final result set (e.g., in the example above, B would be erroneously considered *on-path* even if the attack came from F). Yet, these induced changes cannot "hide" actual on-path ASes from detection. Our graph algorithm on the other hands requires a small modification (shown in Figure 14) to correctly handle induced changes: Whenever the candidate set is reduced, ASes that could show the observed behaviour due to induced changes need to be retained.

To assess the impact of induced route changes, we ran a simulation of our graph-based traceback with these modifications. As shown in Figure 15, neither runtime nor success rate differ significantly compared to the unmodified version. This can be explained because ambiguous ASes are relatively rare: Analyzing the flowgraph used during simulation reveals only 231 ambiguous ASes. We can thus conclude, that induced route changes only have negligible impact on the traceback performance.

```
procedure UPDATEPASSIVE'($\mathcal{P}, r_{\text{passive}}$)
    if $r_{\text{passive}} = \texttt{STOP}$ then
        $\mathcal{C}_{stop} \leftarrow \text{indStop}_{G_A}(\mathcal{P})$
        $\mathcal{C} \leftarrow \mathcal{C} \cap (\text{dominatees}_{G_A}(\mathcal{P}) \cup \mathcal{C}_{stop})$
    else if $r_{\text{passive}} = \texttt{TTL\_CHANGE}$ then
        $\mathcal{C}_{change} \leftarrow \text{indChange}_{G_A}(\mathcal{P})$
        $\mathcal{C} \leftarrow \mathcal{C} \cap (\text{reachable}_{G_A}(\mathcal{P}) \cup \mathcal{C}_{change})$

procedure UPDATEACTIVE'($\mathcal{P}, r_{\text{passive}}, \vec{r}_{\text{active}}$)
    if $r_{\text{passive}} = \texttt{STOP}$ then
        $\mathcal{P}_{\text{inconsistent}} \leftarrow \{X \in \mathcal{P} \mid \vec{r}_{\text{active}}[X] \neq \texttt{STOP}\}$
        $\mathcal{C}_{stop} \leftarrow \text{indStop}_{G_A}(\mathcal{P})$
        $\mathcal{C} \leftarrow \mathcal{C} \setminus (\text{reachable}\, G_A(\mathcal{P}_{\text{inconsistent}}) \setminus \mathcal{C}_{stop})$
    else
        $\mathcal{P}_{\text{inconsistent}} \leftarrow \{X \in \mathcal{P} \mid \vec{r}_{\text{active}}[X] = \texttt{STOP}\}$
        $\mathcal{C}_{change} \leftarrow \text{indChange}_{G_A}(\mathcal{P})$
        $\mathcal{C} \leftarrow \mathcal{C} \setminus (\text{dominatees}_{G_A}(\mathcal{P}_{\text{inconsistent}}) \setminus \mathcal{C}_{change})$
    $\mathcal{P} \leftarrow \mathcal{P} \setminus \mathcal{P}_{\text{inconsistent}}$
```

Figure 14: Flow Graph based traceback algorithm adapted to handle induced route changes

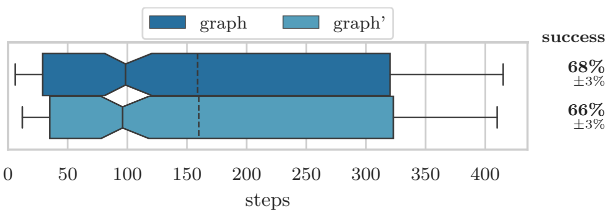


Figure 15: Influence of induced route changes. The modified algorithm is labeled *graph'*.

7.5. Multi-Source Attacks

Amplification attacks are largely launched from single sources [5], [19]. For BGPEEK-A-BOO we thus assume that every attack has a unique source AS. In theory though, amplification attacks could also be launched from multiple colluding sources in different ASes.

If the attack traffic of a multi-source attack can be separated by source (e.g., by different TTL values due to different path lengths), both algorithms of BGPEEK-A-BOO still work as before. Otherwise, successful poisoning of one of the sources can only be measured as a decrease in attack traffic volume. Yet, even in that case, the naive algorithm should be able to find all attack sources.

7.6. Evasion

During a traceback run BGPEEK-A-BOO creates a large number of route advertisements for the probing prefixes. Attackers that are aware of our system may therefore try to evade it by monitoring public BGP data [59], [60] in order to identify and exclude the probing prefixes. While we cannot *prevent* such active evasion attempts, we can at least *detect* if an attacker is evading our system. In this case, we would not observe any attacks from a given adversary at the probe honeypots, but would keep observing them at the control honeypots. Yet, even outside of BGPEEK-A-BOO, amplification honeypots are inherently detectable due to their rate limiting behavior [19].

Besides evasion, attackers might also attempt to deceive BGPEEK-A-BOO by modifying their initial TTL values. However, as shown in Section 6.2.7, BGPEEK-A-BOO still performs well even when ignoring TTL values.

7.7. Ethical Considerations

We took several measures to ensure that our experiments did not impact other systems or lead to instabilities in the BGP. To obtain a real-time estimate of the currently active BGP path, we conducted traceroute runs from RIPE Atlas probes to our measurement system hosted at PEERING. In order to keep the impact on other systems minimal, we only used one-off measurements with the default values defines by RIPE (i.e., packets of 48 bytes, at most 3 packets per destination). For our active ping measurements we ensured to send at most one packet per minute per target on average. At 64 bytes per packet (1Bps), we believe these to have negligible impact.

All experiments that involved sending (poisoned) BGP advertisements were conducted only after consulting PEERING operators. To further ensure that other experiments running at the testbed were not influenced by ours, we used temporary prefixes and a temporary ASN allocated by our regional Internet registry for the purpose of these experiments. We also registered our temporary allocations in the WHOIS database, such that network operators were able to contact us directly. Additionally, we closely monitored network operator mailing lists.

7.7.1. Impact on Legitimate Traffic. Poisoning advertisements can render a prefix temporarily unreachable from parts of the Internet. Therefore, BGPEEK-A-BOO's probing prefixes should host no other systems but honeypot reflectors. As we use BGP Poisoning on these small prefixes only, other prefixes remain unaffected—effectively excluding collateral damage on benign traffic.

8. Related Work

We find related work from two fields of study: The first considers the problem of IP spoofing and traceback, the second is concerned with BGP Poisoning as a primitive for traffic engineering and security applications.

8.1. IP Spoofing and Traceback

IP spoofing and the resulting need for traceback has been an active field of research for many years. One common approach collects (statistical) telemetry data at multiple routers from which the origin of spoofed packets can later be derived [13]–[17]. Another approach lets routers encode path information into the packet itself [61], [62], using additional IP options or unused header fields [7], [61], [63]–[65], advanced encoding schemes [7], [12], [66], or probabilistic techniques [8]–[10], [61], [67] to reduce the per-router overhead. In theory, both approaches could perfectly track the origin of spoofed traffic. However, both require a widespread deployment in routers and the cooperation of multiple ISPs. As some of these techniques have been proposed almost two decades ago, it is clear that this is a inhibiting factor for traceback in practice. In contrast, BGPEEK-A-BOO requires no cooperation of other systems nor changes to existing routers.

Another line of work considers the problem of traceback in the specific case of amplification attacks. Krupp et al. [4], [5] relaxed the traceback problem to finding scanning systems used for attack preparation or re-identifying Booter services responsible for attacks. In contrast, our approach can identify ASes actively participating in the attack without upfront knowledge of the system.

A third line of research considers the broader question of identifying systems that are *capable* of IP spoofing [68]–[71]. While another important step in alleviating the problem of IP spoofing, we consider their work orthogonal to ours: Whereas they find systems that could in principle send spoofed packets, we aim to identify malevolent actors red-handed.

In concurrent work, Fonseca et al. [72] also attempt to identify spoofing sources by varying BGP advertisements from multiple anycast locations. For this, they create 700 different advertisement configurations by selecting subsets of their peers, adding path prepends, and poisoning immediate neighbors. For every configuration they then record through which peer the attack traffic is received, thereby generating a fingerprint of source ASes. In contrast to BGPEEK-A-BOO, their approach necessarily requires 700 probing steps and cannot distinguish ASes that share a common path beyond their immediate neighbors.

8.2. BGP Poisoning

Although only a side-effect of BGP's loop detection mechanism, BGP Poisoning has seen a wide field of applications. In the area of measurement studies, Colitti et al. [73] and Anwar et al. [74] employed BGP Poisoning to supplement the analysis of AS relationships and prefix propagation. Katz-Bassett et al. [25], [26] showed how BGP Poisoning may be used to actively repair routes, and Smith et al. [27] later showed how it can be used to avoid DDoS congested links. Finally, works by Tran et al. [75] and Smith et al. [34] analyze how well BGP Poisoning works in practice through extensive measurements. Yet, to the best of our knowledge, our approach and the concurrent work by Fonseca et al. [72] are the first to leverage BGP Poisoning for DDoS attack traceback.

9. Conclusion

IP spoofing not only enables amplification attacks, but also hides the attackers' true whereabouts. Our system BGPEEK-A-BOO employs a novel traceback approach that shows that BGP Poisoning can be used to track down an attacker's network location—requiring neither the assistance of external parties nor knowing the attacker in advance. We find that our naive algorithm has a median runtime of 549 steps, or just under four days with realistic parameters, thus showing the feasibility of our approach in practice. Our second algorithm leverages a graph model of BGP path propagation built from AS relationship data and manages to reduce this runtime to 98.5 steps, or just over one day, for the same parameters—and in only 29 steps, under five hours, in a quarter of all cases.

Acknowledgment

We would like to thank PEERING for letting us conduct real-world BGP measurements as well as RIPE NCC for a temporary IP-prefix and ASN allocation and the RIPE Atlas platform. We would further like to thank Ethan Katz-Bassett and Italo Cunha for an insightful discussion of an earlier draft of this paper. Finally, we would also like to thank the anonymous reviewers and our shepherd for their valuable feedback.

References

[1] C. Rossow, "Amplification hell: Revisiting network protocols for ddos abuse.," in *NDSS*, 2014.

[2] "NETSCOUT Arbor Confirms 1.7 Tbps DDoS Attack; The Terabit Attack Era Is Upon Us." https://www.netscout.com/blog/asert/netscout-arbor-confirms-17-tbps-ddos-attack-terabit-attack-era. Accessed: 2020-06-18.

[3] "AWS said it mitigated a 2.3 Tbps DDoS attack, the largest ever." https://www.zdnet.com/article/aws-said-it-mitigated-a-2-3-tbps-ddos-attack-the-largest-ever/. Accessed: 2020-06-18.

[4] J. Krupp, M. Backes, and C. Rossow, "Identifying the scan and attack infrastructures behind amplification ddos attacks," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1426–1437, 2016.

[5] J. Krupp, M. Karami, C. Rossow, D. McCoy, and M. Backes, "Linking amplification ddos attacks to booter services," in *International Symposium on Research in Attacks, Intrusions, and Defenses*, pp. 427–449, Springer, 2017.

[6] A. John and T. Sivakumar, "Ddos: Survey of traceback methods," *International Journal of Recent Trends in Engineering*, vol. 1, no. 2, p. 241, 2009.

[7] D. X. Song and A. Perrig, "Advanced and authenticated marking schemes for ip traceback," in *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, vol. 2, pp. 878–886, IEEE, 2001.

[8] B. Duwairi, A. Chakrabarti, and G. Manimaran, "An efficient probabilistic packet marking scheme for ip traceback," in *International Conference on Research in Networking*, pp. 1263–1269, Springer, 2004.

[9] Q. Dong, S. Banerjee, M. Adler, and K. Hirata, "Efficient probabilistic packet marking," in *13TH IEEE International Conference on Network Protocols (ICNP'05)*, pp. 10–pp, IEEE, 2005.

[10] R. Shokri, A. Varshovi, H. Mohammadi, N. Yazdani, and B. Sadeghian, "Ddpm: dynamic deterministic packet marking for ip traceback," in *2006 14th IEEE International Conference on Networks*, vol. 2, pp. 1–6, IEEE, 2006.

[11] A. Belenky and N. Ansari, "On deterministic packet marking," *Computer Networks*, vol. 51, no. 10, pp. 2677–2700, 2007.

[12] Z. Gao and N. Ansari, "A practical and robust inter-domain marking scheme for ip traceback," *Computer Networks*, vol. 51, no. 3, pp. 732–750, 2007.

[13] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer, "Hash-based ip traceback," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 3–14, 2001.

[14] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, B. Schwartz, S. T. Kent, and W. T. Strayer, "Single-packet ip traceback," *IEEE/ACM Transactions on networking*, vol. 10, no. 6, pp. 721–734, 2002.

[15] J. Li, M. Sung, J. Xu, and L. Li, "Large-scale ip traceback in high-speed internet: Practical techniques and theoretical foundation," in *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*, pp. 115–129, IEEE, 2004.

[16] T. Korkmaz, C. Gong, K. Sarac, and S. G. Dykes, "Single packet ip traceback in as-level partial deployment scenario," *International Journal of Security and Networks*, vol. 2, no. 1-2, pp. 95–108, 2007.

[17] M. Sung, J. Xu, J. Li, and L. Li, "Large-scale ip traceback in high-speed internet: practical techniques and information-theoretic foundation," *IEEE/ACM Transactions on Networking*, vol. 16, no. 6, pp. 1253–1266, 2008.

[18] "ISP Cut off From Internet After Security Concerns." https://www.pcworld.com/article/153734/mccolo_isp_security.html. Accessed: 2021-02-03.

[19] L. Krämer, J. Krupp, D. Makita, T. Nishizoe, T. Koide, K. Yoshioka, and C. Rossow, "Amppot: Monitoring and defending against amplification ddos attacks," in *International Symposium on Recent Advances in Intrusion Detection*, pp. 615–636, Springer, 2015.

[20] B. Schlinker, T. Arnold, I. Cunha, and E. Katz-Bassett, "Peering: Virtualizing bgp at the edge for research," in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, pp. 51–67, 2019.

[21] "RIPE Atlas." https://atlas.ripe.net. Accessed: 2019-12-01 - 2019-12-03.

[22] Y. Rekhter, S. Hares, and T. Li, "A Border Gateway Protocol 4 (BGP-4)," Tech. Rep. 4271, Jan. 2006.

[23] M. Lepinski and S. Kent, "An Infrastructure to Support Secure Internet Routing," Tech. Rep. 6480, Feb. 2012.

[24] M. Lepinski, D. Kong, and S. Kent, "A Profile for Route Origin Authorizations (ROAs)," Tech. Rep. 6482, Feb. 2012.

[25] E. Katz-Bassett, D. R. Choffnes, Í. Cunha, C. Scott, T. Anderson, and A. Krishnamurthy, "Machiavellian routing: improving internet availability with bgp poisoning," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, pp. 1–6, 2011.

[26] E. Katz-Bassett, C. Scott, D. R. Choffnes, Í. Cunha, V. Valancius, N. Feamster, H. V. Madhyastha, T. Anderson, and A. Krishnamurthy, "Lifeguard: Practical repair of persistent route failures," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 395–406, 2012.

[27] J. M. Smith and M. Schuchard, "Routing around congestion: Defeating ddos attacks and adverse network conditions via reactive bgp routing," in *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 599–617, IEEE, 2018.

[28] "The search engine for the Internet of Things." https://www.shodan.io/. Accessed: 2020-06-18.

[29] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed internet routing convergence," *ACM SIGCOMM Computer Communication Review*, vol. 30, no. 4, pp. 175–187, 2000.

[30] "AS65000 BGP Routing Table Analysis Report." https://bgp.potaroo.net/as2.0/. Accessed: 2020-04-29.

[31] Y. Zhang, R. Oliveira, H. Zhang, and L. Zhang, "Quantifying the pitfalls of traceroute in as connectivity inference," in *International Conference on Passive and Active Network Measurement*, pp. 91–100, Springer, 2010.

[32] Y. Hyun, A. Broido, *et al.*, "Traceroute and bgp as path incongruities," tech. rep., Cooperative Association for Internet Data Analysis (CAIDA), 2003.

[33] L. Gao, "On inferring autonomous system relationships in the internet," *IEEE/ACM Transactions on Networking (ToN)*, vol. 9, no. 6, pp. 733–745, 2001.

[34] J. M. Smith, K. Birkeland, T. McDaniel, and M. Schuchard, "Withdrawing the bgp re-routing curtain," in *Network and Distributed System Security Symposium (NDSS)*, 2020.

[35] M. Schuchard, A. Mohaisen, D. Foo Kune, N. Hopper, Y. Kim, and E. Y. Vasserman, "Losing control of the internet: using the data plane to attack the control plane," in *Proceedings of the 17th ACM conference on Computer and communications security*, pp. 726–728, 2010.

[36] M. Schuchard, J. Geddes, C. Thompson, and N. Hopper, "Routing around decoys," in *Proceedings of the 2012 ACM conference on Computer and communications security*, pp. 85–96, 2012.

[37] "The CAIDA AS Relationships Dataset, Jun 2019 - Jan 2020." http://www.caida.org/data/active/as-relationships/.

[38] R. Bush, O. Maennel, M. Roughan, and S. Uhlig, "Internet optometry: assessing the broken glasses in internet reachability," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, pp. 242–253, 2009.

[39] "Team Cymru IP to ASN Lookup v1.0." https://whois.cymru.com/. Accessed: 2019-12-01 - 2019-12-03.

[40] "RIPEstat Data API." https://stat.ripe.net/docs/data_api. Accessed: 2020-01-21.

[41] C. Villamizar, R. Chandra, and D. R. Govindan, "BGP Route Flap Damping," Tech. Rep. 2439, Nov. 1998.

[42] Z. M. Mao, R. Govindan, G. Varghese, and R. H. Katz, "Route flap damping exacerbates internet routing convergence," in *ACM SIGCOMM Computer Communication Review*, vol. 32, pp. 221–233, ACM, 2002.

[43] P. Smith and C. Panigl, "RIPE Routing Working Group Recommendations On Route-flap Damping." RIPE 378, May 2006.

[44] R. Bush, C. Pelsser, M. Kuhne, O. Maennel, P. Mohapatra, K. Patel, and R. Evans, "RIPE Routing Working Group Recommendations on Route Flap Damping." RIPE 580, Jan. 2013.

[45] C. Pelsser, R. Bush, K. Patel, P. Mohapatra, and O. Maennel, "Making Route Flap Damping Usable," Tech. Rep. 7196, May 2014.

[46] C. Pelsser, O. Maennel, P. Mohapatra, R. Bush, and K. Patel, "Route flap damping made usable," in *International Conference on Passive and Active Network Measurement*, pp. 143–152, Springer, 2011.

[47] "Cisco IOS IP Routing: BGP Command Reference." https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/iproute_bgp/command/irg-cr-book/bgp-a1.html#wp3674090369. Accessed: 2020-01-08.

[48] W. Mühlbauer, A. Feldmann, O. Maennel, M. Roughan, and S. Uhlig, "Building an as-topology model that captures route diversity," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4, pp. 195–206, 2006.

[49] V. Giotsas, M. Luckie, B. Huffaker, and K. Claffy, "Inferring complex as relationships," in *Proceedings of the 2014 Conference on Internet Measurement Conference*, pp. 23–30, 2014.

[50] V. Giotsas and S. Zhou, "Valley-free violation in internet routing—analysis based on bgp community data," in *2012 IEEE International Conference on Communications (ICC)*, pp. 1193–1197, IEEE, 2012.

[51] L. Subramanian, S. Agarwal, J. Rexford, and R. H. Katz, "Characterizing the internet hierarchy from multiple vantage points," in *Proceedings. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 2, pp. 618–627, IEEE, 2002.

[52] G. Di Battista, M. Patrignani, and M. Pizzonia, "Computing the types of the relationships between autonomous systems," in *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No. 03CH37428)*, vol. 1, pp. 156–165, IEEE, 2003.

[53] T. Erlebach, A. Hall, and T. Schank, "Classifying customer-provider relationships in the internet," *TIK-Report*, vol. 145, 2002.

[54] J. Xia and L. Gao, "On the evaluation of as relationship inferences [internet reachability/traffic flow applications]," in *IEEE Global Telecommunications Conference, 2004. GLOBECOM'04.*, vol. 3, pp. 1373–1377, IEEE, 2004.

[55] X. Dimitropoulos, D. Krioukov, B. Huffaker, G. Riley, *et al.*, "Inferring as relationships: Dead end or lively beginning?," in *International Workshop on Experimental and Efficient Algorithms*, pp. 113–125, Springer, 2005.

[56] X. Dimitropoulos, D. Krioukov, M. Fomenkov, B. Huffaker, Y. Hyun, K. Claffy, and G. Riley, "As relationships: Inference and validation," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 1, pp. 29–40, 2007.

[57] B. Hummel and S. Kosub, "Acyclic type-of-relationship problems on the internet: an experimental analysis," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pp. 221–226, 2007.

[58] M. Luckie, B. Huffaker, A. Dhamdhere, V. Giotsas, *et al.*, "As relationships, customer cones, and validation," in *Proceedings of the 2013 conference on Internet measurement conference*, pp. 243–256, ACM, 2013.

[59] "University of Oregon Route Views Archive Project." http://routeviews.org/. Accessed: 2020-11-02.

[60] "RIPE Routing Information Service (RIS)." http://www.ripe.net/data-tools/stats/ris. Accessed: 2020-11-02.

[61] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Practical network support for ip traceback," in *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pp. 295–306, 2000.

[62] T. W. Doeppner, P. N. Klein, and A. Koyfman, "Using router stamping to identify the source of ip packets," in *Proceedings of the 7th ACM Conference on Computer and Communications Security*, pp. 184–189, 2000.

[63] D. Dean, M. Franklin, and A. Stubblefield, "An algebraic approach to ip traceback," *ACM Transactions on Information and System Security (TISSEC)*, vol. 5, no. 2, pp. 119–137, 2002.

[64] A. Yaar, A. Perrig, and D. Song, "Stackpi: New packet marking and filtering mechanisms for ddos and ip spoofing defense," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 10, pp. 1853–1863, 2006.

[65] R. Chen, J.-M. Park, and R. Marchany, "A divide-and-conquer strategy for thwarting distributed denial-of-service attacks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 5, pp. 577–588, 2007.

[66] A. Yaar, A. Perrig, and D. Song, "Pi: A path identification mechanism to defend against ddos attacks," in *2003 Symposium on Security and Privacy, 2003.*, pp. 93–107, IEEE, 2003.

[67] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Network support for ip traceback," *IEEE/ACM transactions on networking*, vol. 9, no. 3, pp. 226–237, 2001.

[68] R. Beverly and S. Bauer, "The spoofer project: Inferring the extent of source address filtering on the internet," in *Usenix Sruti*, vol. 5, pp. 53–59, 2005.

[69] R. Beverly, R. Koga, and K. Claffy, "Initial longitudinal analysis of ip source spoofing capability on the internet," *Internet Society*, p. 313, 2013.

[70] M. Kührer, T. Hupperich, C. Rossow, and T. Holz, "Exit from hell? reducing the impact of amplification ddos attacks," in *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pp. 111–125, 2014.

[71] M. Luckie, R. Beverly, R. Koga, K. Keys, J. A. Kroll, and k. claffy, "Network hygiene, incentives, and regulation: Deployment of source address validation in the internet," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 465–480, 2019.

[72] O. Fonseca, Í. Cunha, E. Fazzion, W. Meira, B. Junior, R. A. Ferreira, and E. Katz-Bassett, "Tracking down sources of spoofed ip packets," in *2020 IFIP Networking Conference (Networking)*, pp. 208–216, IEEE, 2020.

[73] L. Colitti, G. Di Battista, M. Patrignani, M. Pizzonia, and M. Rimondini, "Investigating prefix propagation through active bgp probing," *Microprocessors and Microsystems*, vol. 31, no. 7, pp. 460–474, 2007.

[74] R. Anwar, H. Niaz, D. Choffnes, Í. Cunha, P. Gill, and E. Katz-Bassett, "Investigating interdomain routing policies in the wild," in *Proceedings of the 2015 Internet Measurement Conference*, pp. 71–77, 2015.

[75] M. Tran, M. S. Kang, H.-C. Hsiao, W.-H. Chiang, S.-P. Tung, and Y.-S. Wang, "On the feasibility of rerouting-based ddos defenses," in *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 1169–1184, IEEE, 2019.

Appendix A. Correctness of AS Flow Graphs

Theorem 1. *Let $\Pi = (X_1, \dots, X_n) \in \mathrm{AS}^n$ be a valley-free path from AS X_1 to AS X_n and $G = (V, E)$ the flow graph constructed according to Section 5.1. Then there exists a path $\pi = (x_1 = u_{X_1}, \dots, x_m = c_{X_n}) \in V^m$ in G.*

Proof. Since $\Pi = (X_1, \dots, X_n)$ is a valley-free path, it can be split into three (possibly empty) parts: an "uphill" prefix, a single "peak" peer-to-peer link, and a "downhill" suffix. Formally, $\exists k, l, 1 \leq k \leq l \leq k+1 \leq n$ s.t.:

1) $\forall 1 \leq i < k : (X_i, X_{i+1}) \in \mathrm{CP}$
2) $k < l \implies (X_k, X_l) \in \mathrm{P2P}$
3) $\forall l \leq i < n : (X_{i+1}, X_i) \in \mathrm{CP}$

For the first part, we can find a path in G as by construction it holds that $\forall 1 \leq i < k : (u_{X_i}, u_{X_{i+1}}) \in E$. Likewise, for the last part, since $(X_{i+1}, X_i) \in \mathrm{CP}$ it holds that $\forall l \leq i < n : (c_{X_i}, c_{X_{i+1}}) \in E$. If the path has a peer-to-peer link, i.e., $k < l$, then $(u_{X_k}, c_{X_l}) \in E$. Otherwise, it holds that $X_k = X_l$ and thus $(u_{X_k}, c_{X_l}) = (u_{X_k}, c_{X_k}) \in E$. Therefore, the path $\pi = (u_{X_1}, \dots, u_{X_k}, c_{X_l}, \dots, c_{X_n})$ is in G and satisfies the requirements. □

Theorem 2. *Let $G = (V, E)$ be the flow graph constructed according to Section 5.1 and $\pi = (x_1, \dots, x_m) \in V^m$ be a path in G. Then there exists a valley-free path $\Pi = (X_1 = \mathrm{asn}(x_1), \dots, X_n = \mathrm{asn}(x_m)) \in \mathrm{AS}^n$.*

Proof. W.l.o.g. assume that $x_1 = u_{X_1}$ and $x_m = c_{X_n}$. Since all edges in E are of the form (u_A, u_B), (c_A, c_B), or (u_A, c_B) the path can only have one transition from "unconstrained" to "constrained" nodes, i.e., $\exists 1 \leq i < n$ such that

1) $\forall 1 \leq j \leq i : x_j \in \{u_A | A \in \mathrm{AS}\}$
2) $\forall i < j \leq n : x_j \in \{c_A | A \in \mathrm{AS}\}$

Therefore, for $j < i$ it holds that $(\mathrm{asn}(x_j), \mathrm{asn}(x_{j+1})) \in \mathrm{CP}$, as otherwise there would be no edge in G between them. Likewise, for $j > i$ it holds that $(\mathrm{asn}(x_{j+1}), \mathrm{asn}(x_j)) \in \mathrm{CP}$. Finally, for the edge $(x_i, x_{i+1}) = (u_{X_k}, c_{X_l})$ it must hold that either $X_k = X_l$ or that $(X_k, X_l) \in \mathrm{P2P}$. Therefore, $\Pi = (\mathrm{asn}(x_1), \dots, \mathrm{asn}(x_m))$ follows the definition of a valley-free path, with the exception that it may still contain loops. However, if Π contains a loop, i.e., $\exists 1 \leq i < j \leq n$ s.t. $\Pi = (X_1, \dots, X_i, \dots, X_j, \dots X_n)$ with $X_i = X_j$, one can easily construct a loop-free path $\Pi' = (X_1, \dots, X_i, \dots X_n)$ by omitting positions $i+1$ through j. Note that removal of loops does not affect the path's endpoints, and thus the resulting AS path Π' is a valley-free path from $\mathrm{asn}(x_1)$ to $\mathrm{asn}(x_m)$. □