Detection and Recovery from the Oblivious Engineer Attack

by

Jennifer Joyce Mulligan

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2002

Author .....
Department of Electrical Engineering and Computer Science
September 9, 2002

Certified by .....
Ronald L. Rivest
Andrew and Erna Viterbi Professor of
Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Detection and Recovery from the Oblivious Engineer Attack

by

Jennifer Joyce Mulligan

Submitted to the Department of Electrical Engineering and Computer Science
on September 9, 2002, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

The weak security of the Border Gateway Protocol (BGP) has been a known issue and has had several solutions proposed. The leading solution of the Secure BGP protocol has been slow to be acceptably refined or adopted. Instead, here I consider a somewhat smaller problem of a specific attack on the network and propose a solution to the attack that may impact other insecurities of BGP as well. The oblivious engineer attack is one in which a router will allow all BGP traffic to pass and be processed correctly while dropping all other traffic along that route. Part of the network has been effectively cut off, but BGP will fail to notice because its own traffic is still succeeding.

The proposed solution involves more than just the routers in the exterior routing protocols. Instead, the end hosts participate by alerting the BGP speakers to connection failures that they detect. In turn the BGP speaker may investigate the complaint and determine if the link is indeed failed. If it has failed, then the BGP speaker will seek out another working route from other available routes through which to pass the traffic. In this way the end BGP speakers will learn of the problem and correct it themselves, if possible, by rerouting the traffic through a different path.

The attack and solution were simulated using SSFNet. The results were as expected as the solution will avert the attack by finding another route. The third connection attempt between the host and the server will succeed and progress normally, with a practically negligible impact on the traffic of the network as a whole. The first two connection attempts are needed to alert both ends of the connection to the failure, assuming that the failed router is on both paths from one node to the other.

Thesis Supervisor: Ronald L. Rivest
Title: Andrew and Erna Viterbi Professor of
Electrical Engineering and Computer Science

Acknowledgments

I would like to thank my advisor, Ronald Rivest for his assistance in producing this work. I would also like to thank BJ Premore for his help in understanding and working with SSFnet, specifically the BGP modules. I am also very grateful to my fiancee Jordi Albornoz for helping to keep me on task and developing my Java programming skills. My parents have also been a great help to me by commenting on and helping me revise my writing.

Contents

List of Figures

Chapter 1

Introduction

As the Internet has become a public entity, a forum for business and an essential communication medium, the more attractive it is to attack. According to the CERT Coordination Center (originally the Computer Emergency Response Team), "one of the most recent and disturbing trends we have seen is an increase in intruder compromise and use of routers."[3] Attacks on the routers of the network attack the very basis of the Internet infrastructure. Each router is responsible for sending data correctly around the Internet, delivering it to its final destination. Business depends heavily upon a functioning network which makes it a likely target for those looking to harm our capitalist society. Such attacks come at a great cost. For example, in the year 2000, many prominent web sites such as *Yahoo!*, *eBay*, and others were afflicted with a distributed denial of service attack costing at least 1.2 billion dollars[5]. The protection we seek may be available but there is an inherent conflict between the open communication and association we all desire and the need to secure such communications. For example, one wants to be able to conduct banking transactions at any hour of the day over the web but still prevent unauthorized access.

A specifically vulnerable aspect of the Internet is the routing infrastructure. In other words, the mechanism through which all the messages find their way across the Internet. All data sent through the Internet is broken into small segments called packets. These packets flow across the network separately. When they arrive at their final destination they are reassembled in the proper order. Each packet not only

contains data, but also labels that describe the data. Examples of packet labels are the source of the data, the final destination, and the protocol to which the packet belongs. From these labels, the computers can tell where to send the packet. For more details and background about computer networks see author [13].

In order to get from their source to their destination, the packets must be successively sent from one computer to the next. The computers that perform these functions are called routers, because they route packets. At each step, each router must know in what direction to send the packet for best results. Therefore, they run routing protocols which distribute information about both the layout of the network and about what path is the shortest from one destination to another. All routes are learned in good faith from one's neighbor with little security or verification. This is akin to an investor listening exclusively to his three brokers about the state of the stock market and what he should do. If the advisors do their job properly, the investor should be very successful, but if any are corrupted he may lose any money that was invested based on the evil investor's advice. In this paper we look to provide more security to the primary Internet routing protocol, BGP, to keep it running smoothly, delivering traffic effectively and efficiently.

Here we consider an attack, here entitled The Oblivious Engineer Attack, developed to work against the routing protocols and the structure of the Internet, i.e. to stop packet delivery. The routing protocol BGP, along with its successor Secure BGP, has a key weakness that may be exploited in an attack. The protocols assume that if their own traffic is delivered, then all traffic over the link is delivered and thus the network is running fine in their view. An attack can easily be developed to separate routing traffic from all other traffic and only deliver the routing traffic. This effectively destroys the connection, but no currently known automated method will discover such failure. Any email sent through that router will be dropped and never delivered. Any web request sent through the failing router will also be dropped, with a message of "unable to connect to server" returned to the user. Therefore we look to design a defense against this attack that is more automatic. One approach to securing BGP that has already been designed is Secure BGP. It provides additional authenti-

cation and encryption through an elaborate public key infrastructure. Unfortunately it still is susceptible to the Oblivious Engineer attack.

The defense approach is straight-forward, designed to use minimal resources, and not add any noticeable traffic to the Internet. It is important to keep the mechanism simple enough to be adopted, and be reasonably effective while being incrementally deployed on more and more routers throughout the network. This allows early adopters to benefit from the technology immediately. They won't have to wait until the majority of the Internet has also implemented the solution. We also look to place the responsibility of the effort and its cost directly on the same systems that will derive the advantage from such actions. Following these guidelines, this proposal seeks to provide countermeasures to the Oblivious Engineer attack and bring more flexibility into the routing protocol by using live probing and feedback about the network. The feedback comes from the individual clients running on hosts machines, and the probing is conducted by the machines running BGP exploring alternative options to a broken path.

After developing the attack and solution, we then wrote the attack and solution in a simulation. The simulation allowed us to give a proof of concept, that the solution rerouted around the attack, as well as examine other details that we had not yet considered. It brought to light new ideas and possible ways of solving problems. Implementation difficulties also arose giving a more realistic view of the work needed to create the attack, and the solution.

In chapter 2 we discuss the background information necessary to understand the problem and the solution. It covers the packet delivery mechanism of the network, how the network is viewed from an administrative standpoint, and the different types of routing algorithms with specific emphasis on BGP, the Border Gateway Protocol. It also discusses the simulation program used for the results in the paper and the simulations nuances. Chapter 3 describes the Oblivious Engineer Attack, the implementation and its possible consequences. Chapter 4 covers the solution mechanism entitled Extended BGP, how it works, how it was implemented within the simulation program, and what other options might be considered for an implementation.

The details of the simulation run and the results from the simulation are explained in chapter 5. We then summarize the weaknesses of the approach and what future work would be beneficial to expand and test the solution. Appendix A contains the detailed network specification used in the simulation, the precise simulation output and the code used to implement the Oblivious Engineer attack.

Chapter 2

Background

To understand the mechanisms used in the Oblivious Engineer Routing Attack one must understand the basics of routing in the Internet and how it all fits together. In this chapter we cover all the background material needed to understand the attack and the proposed solution. This includes a description of routing in general as well as the types of routing protocols and the details of one particular routing protocol, BGP.

2.1 Moving Data Through the Internet

To get data, such as an email, from your computer to your friend's computer, many things need to happen. At your computer the email is broken up into smaller pieces to be sent across the network. This is because each part of the network has rules concerning the maximum data size that it will transfer. These rules allow the network to function efficiently and with minimal errors. The pieces of data are then sent to your computer's default router. The default router is the one router specified to your computer to be the best route out of the local network. In fact, at all points between your computer and your friends computer all of the data is handled by routers. A router is defined as a computer whose job it is to pass data along to other computers when appropriate. Your end host computer would not pass data through it, and likely only has one connection to the network. A router on the other hand has at least two

connections available to it. These connections to the network are called the interfaces of the router. As your email travels through the network, a sequence of routers will send the data coming in from one of their interfaces, and out another. Most routers have more than two interfaces, and so when data arrives that is not addressed to them, they need to decide which outgoing link makes the most sense. Passing data along like this is called forwarding. The decision of which interface to use is the basic functionality of routing in the Internet.

2.1.1 Forwarding Tables

At its very basis, the Internet is a packet based system. That means that each packet of data is forwarded individually as it arrives based only on information within the packet. Forwarding packets along the proper interface requires that the router know which interface leads a packet closer on its path to its final destination. The router does this by consulting its forwarding table. The forwarding table consists of a list of destination IP addresses and the corresponding outgoing link that is the most appropriate next hop along the path. IP addresses are the naming system used in the Internet to uniquely identify each node. Additional information may also be stored but this is the basic required information. When a packet arrives, its destination IP address is read from the packet header and matched to the best entry in the forwarding table. The entry lists which interface should be used and using this information the router sends the packet on that link. The forwarding table is an efficient and minimal state mechanism to forward packets through the Internet to their final destination. Bit comparisons of IP addresses can be done in hardware and so they are very fast.

In order for a router to have a forwarding table, it must get the information about how all other computers are connected together. This information is gathered using a routing protocol which is run by all routers throughout the Internet.

2.2 Routing Algorithms

In its most general form, a routing algorithm is a distributed algorithm that transmits information about how each computer is connected. A distributed algorithm is one in which the computation is done on several different computers with the output being the set of the outputs on the individual machines. The information is used by each router to find the shortest path to all other points. In a more perfect world, a routing algorithm would only need to be run each time a new computer is added or an old one removed. In a realistic world routers and links between them can be unstable and may break, reboot, or drop packets and then later reconnect. So a routing protocol needs to be running at all times to keep track of such errors and recoveries, as well as new additions or removals. It needs to be flexible and adjust to changes by figuring out new ways to route traffic when its currently used routes have failed.

Two main types of routing algorithms have been developed, each with their own strengths and weaknesses. The primary differences between them are the amount of knowledge each individual router attains about the network layout and the information distribution method. In a link-state protocol each host learns the entire layout of the network, where as in a distance-vector protocol each host learns only the distance and next hop used to get to each destination.

2.2.1 Link-State Protocol

A Link-State protocol is a routing protocol that distributes enough information for each node (computer) to know the entire layout of the network and analyze the network for itself. Every computer sends out what its part of the network looks like, namely it sends out its name (IP address) and a list of the names of all of its neighbors. The message is propagated throughout the entire network. Propagation is done by each node sending any new routing message out all outgoing paths other than the one that the message arrived on. Otherwise, if the router has already seen the message it does nothing. In this way all nodes in the network get all of the messages of all other nodes and can piece together the picture of the entire network. Each node then

decides its chosen routes based on shortest path algorithms such as Dijkstra's[2] or individual policy.

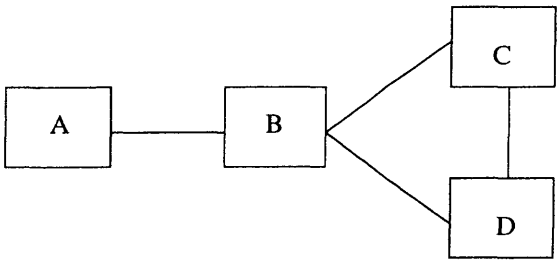
This method is highly effective for allowing each node to know precisely where its traffic will be traveling and picking its most preferable route. Given a small number of computers, there won't be a lot of routing information traffic, but this method does not scale well to larger networks because each router will receive a lot of traffic. It will receive at least one message for every node in the network, often receiving packets more than once. It is also inefficient for a large number of nodes because each router would have to remember at least as many packets as there are nodes in a network to know if it has already seen the packet. Therefore, we look for a different solution that uses less bandwidth for larger networks but still distributing enough information to route packets.

2.2.2 Distance-Vector Protocol

A Distance-Vector protocol is better suited for larger networks as it scales well. It functions on the basic principle that the most important characteristic of the path taken is the overall length of the path. The shortest path is preferred. The protocol is similar to link-state protocols in that each node sends a list of its neighbors to all of its neighbors. Instead of only sending the names they also send a "hop count". This lets the receiver know how many hops away the other computer is. For instance, a direct neighbor has a hop count of 1 because there is only one link between them. Instead of forwarding these messages, each node then only sends on any new information that provides a new shortest route. It also adds 1 to the hop count before forwarding because the path length has now increased. A notable difference is that each node only talks to its neighbors for information. In a link-state protocol, a single packet may traverse the entire network, being read at each hop along the way. When the protocol has stabilized (if the network is unchanged) each node will know how far away every other node is and the next hop that it should use to get to it (it was the neighbor who sent them the announcement of the path). While this method will scale with larger numbers of computers and still remain efficient, it gives each node

less information and less control about how its traffic is sent. It gains its efficiency by sending fewer packets and only those are that necessary. Each packet also contains less information and so is smaller.

Figure 2-1 is an example of a Distance Vector protocol including the forwarding tables developed at each node. The list of all messages sent is done in rounds to give a general timeline of when the messages would be sent. At each round the nodes learn of other nodes that are further and further away. Before the algorithm begins, each node knows only of itself. Then the first round ends with all nodes knowing the names of their direct neighbors. The second round then gives each node information about all nodes that are two hops away. Finally the third round distributes the last bit of information not yet shared.



Format: (X – #) is (destination – distance)

Round One Messages:
A sends (A – 1) to B
B sends (B – 1) to A, C, D
C sends (C – 1) to B, D
D sends (D – 1) to B, C

Round Two Messages:
B sends (C – 2), (D – 2) to A
B sends (A – 2), (D – 2) to C
B sends (A – 2), (C – 2) to D
C sends (B – 2) to D
C sends (D – 2) to B
D sends (C – 2) to B
D sends (B – 2) to C

Round Three Messages:
C sends (A – 3) to D
D sends (A – 3) to C

Forwarding Tables:
Destination – Next Hop – Distance

	A	B	C	D
Initial Table	A – A – 0	B – B – 0	C – C – 0	D – D – 0
After 1st Rnd.	A – A – 0 B – B – 1	B – B – 0 A – A – 1 C – C – 1 D – D – 1	C – C – 0 B – B – 1 D – D – 1	D – D – 0 B – B – 1 C – C – 1
After 2nd Rnd.	A – A – 0 B – B – 1 C – B – 2 D – B – 2	B – B – 0 A – A – 1 C – C – 1 D – D – 1	C – C – 0 B – B – 1 D – D – 1 A – B – 2	D – D – 0 B – B – 1 C – C – 1 A – B – 2

The third round of updates makes no changes to the forwarding tables.

Figure 2-1: Distance Vector Algorithm and Forwarding Table Example

2.2.3 Exterior vs. Interior Routing Protocols

Both of the above routing protocols serve a distinct purpose in the current Internet. Link-state protocols work for smaller groups, and Distance-Vector is effective in a larger group. The Internet has been separated into such divisions since it is a large number of small groups of computers. Each small group is called an autonomous system or AS. A centralized authority gives each AS an AS number (ASN) that uniquely identifies it. The AS divisions are made based on real world divisions such as company domains or ISPs. Within the AS, link-state protocols are often run to distribute the local routing information because they are small groups. The general term used for all such protocols is Interior Routing Protocol because it is run in the interior of the AS. Alternatively, each AS must learn to route traffic from one to another and does so by using an exterior routing protocol. As each protocol runs, the information is added into the forwarding table so that any incoming packet will be sent in the correct direction, and the routers running both protocols share information through both methods.

2.3 Border Gateway Protocol

The Border Gateway Protocol (BGP)[11] is the exterior routing protocol of the Internet. It is used to distribute routing information between the different autonomous systems. It is not strictly a distance vector protocol or a link-state protocol, instead it is a conglomeration of the two. The algorithm is primarily a distance-vector algorithm, distributing a limited amount of information only to the direct neighbors. It expands upon the distance-vector protocol by identifying the actual nodes on a path like a link-state protocol. Instead of sending just the distance to the destination, it also sends what ASs the traffic will travel through.

In theory, BGP works as described above although there are many other details that are important about how BGP works in practice. Each autonomous system has at least one and possibly more BGP speakers. A BGP speaker is none other than a router that participates in the BGP protocol for the AS. Together the BGP speakers

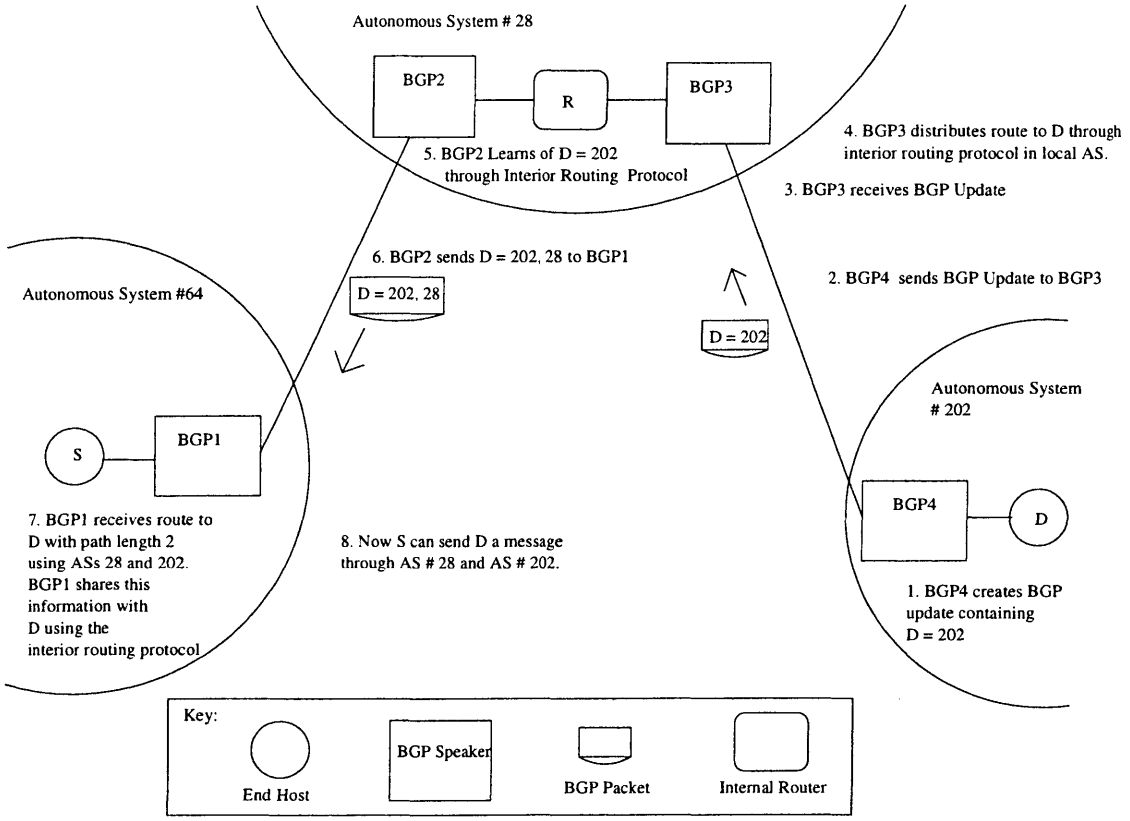


Figure 2-2: BGP Route Distribution Example

control all traffic entering and leaving the AS. Each BGP speaker is directly linked to a BGP speaker in its neighbor AS. Over this wire, they share BGP information using TCP [9]. Conversely the BGP speaker must also share the routing information it has learned about foreign ASs with its own local AS using an interior routing protocol.

An example of BGP performing route distribution is demonstrated in Figure 2-2. In this example, the path to D is shared with S after many steps. To follow the example read from step 1 through step 8 in order.

The security of BGP is weak and leaves it open to spoofing and attack. No authentication is used currently other than the authentication the physical wire provides between the two BGP speakers. An intruder could imitate the real BGP speaker. There is also no verifiability of any routes. Any router could claim to have any route it wished. For instance a spy in a rogue country could claim to be one hop away from the CIA, and as this information is distributed messages meant for the CIA would be sent over the spy's network. The spy could read all of the messages and then pass them on without the CIA knowing any better. In addition, BGP is vulnerable to any typical TCP attacks such as connection hijacking and man-in-the-middle attacks. When BGP was first developed, this was not a large concern as the Internet was generally used by the military and academics who were not looking to break the system. As the Internet has grown, businesses rely on the Internet and it is more important that it remain stable. Thus a new version of BGP has been proposed called Secure BGP to handle some of these security risks.

2.3.1 Secure Border Gateway Protocol

The Secure Border Gateway Protocol [1] has been developed in response to the increasing security threats on the Internet. We have seen many different types of attacks and one that destroys the routing infrastructure of the Internet could cause great damage. Secure BGP is based on BGP but uses a public key infrastructure to provide protection against impersonators and liars. The ideas are loosely based upon work previously done by Radia Perlman[6]. Every BGP speaker is given a public and private key so that it can sign messages, proving that it wrote the message. First, the keys are used to set up a secure the connections between BGP speakers with IPsec [4] using at least the authentication options if not also the encryption ones. Then the security risks are those of IPsec instead of just a TCP connection. Second, the keys are used to sign the list of ASs that show the path that is used to the final destination. At each hop along the way, the AS signs the whole list including itself and the next hop so that it is linked to the expected next hop cryptographically. Then we have a complete signature chain the entire way along the path from current host to final

destination which can be verified for its accuracy.

The normal methods of attacking BGP are generally foiled by S-BGP assuming the public key infrastructure remains secure. If a router in the middle were to try to cheat, it couldn't forge the path of nodes before it because it doesn't know the keys. If an evil node wanted to pretend there were additional nodes after it in the chain it could not do this either. The next real hop in the chain would recognize that the final hop listed in the path is not a node that it is connected to. Therefore the final hop listed could not have legitimately sent the advertisement. One flaw still remains, though, if two routers collude and both lie together. They could send the packets long distance across the network between each other and appropriately sign the path pretending that they are directly connected together when they are really many hops away. This type of attack is particularly difficult to protect against because of the collusion. It is hard to know if those two nodes are connected or not as no one else can check the veracity of their statement except themselves. Overall, the protection offered through S-BGP protects against the majority of attacks and most importantly it protects against the simplest to implement attacks.

Although S-BGP has been developed and in the works for many years, it has yet to be used beyond the research world and reference implementations. It uses a complicated structure of certificates and signatures which adds overhead to the entire protocol. It is a strong protocol with strong protection and may not have been adopted yet because of its large structure and the changes it would bring to the Internet. S-BGP would not be very effective if only implemented by a few routers, but instead needs the majority of the routers to participate for full protection. It can be incrementally deployed which is an absolute necessity for any new protocol today, and groups are working to refine it and see it finally be deployed.

Chapter 3

Oblivious Engineer Attack

The Oblivious Engineer Attack is one which we developed after considering the basic weaknesses of BGP and the routing architecture. It is simplistic in nature but effective at disabling the network. We discuss the details of the attack including the idea for the name, information about the implementation in SSFNet, and the possible damage that such an attack could inflict. The attack can affect either a router, or the link inbetween two BGP speakers.

3.1 Details of the Attack

The Oblivious Engineer Attack takes advantage of a very basic assumption made by both BGP and Secure BGP. They both assume that if their own traffic is being received, then all traffic must be passing through with no trouble as well. The attack takes advantage of this by letting all BGP traffic pass while dropping all other traffic, cutting off part of the network. For instance, all email, HTTP requests, and FTP transfers would be dropped. There are multiple ways this could be accomplished. First, BGP traffic could be individually identified by reading each packet as it arrives and dropping any non-BGP or non-S-BGP packets. The approach may take a great amount of computation power because the router would have to look at the packet contents, not just the packet header. A simpler but more coarse grained approach is to drop all traffic that does not originate at one BGP speaker and have a destination

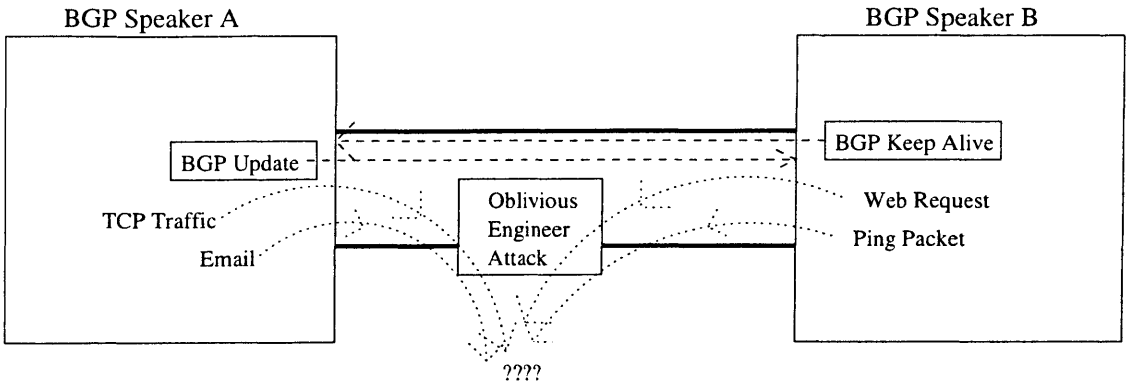


Figure 3-1: Oblivious Engineer Attack

of the other. The router needs only to look at packet headers to do this. Then some other protocol's packets might get through as well as BGP, but the vast majority of traffic handled is not addressed to a BGP speaker and can be dropped. In effect, part of the network has been cut off, as far as transmitting normal network traffic other than routing information , but no basic network protocols will detect such a failure. BGP will see its own traffic pass through and find no failure. It will require human intervention to notice the loss of traffic, detect where the problem is, and intervene to fix it.

Figure 3-1 depicts the attack by an intruder on the wire inbetween two BGP speakers. The wire between them is enlarged to show the passage of the BGP traffic while all other traffic has been deleted from the wire.

3.1.1 Where Did The Name Come From?

The idea for the name of the Oblivious Engineer attack came from a tale of a train engineer. This Oblivious Engineer conducts his train from one destination to the next, without realizing that his cargo has been unlatched from the engine and stolen by robbers. The engine, the control mechanism, functions normally while the cargo, the data, is lost. Similarly, BGP remains oblivious to the attack on the network and delivers its traffic believing that everything is okay.

3.2 Implementation Details

In order to simulate such an attack, we changed the implementation of IP [8] in the network stack of the simulation router. We only needed to add 6 lines of code in the packet processing routine of IP that silently and swiftly dropped non-BGP traffic without passing it on. The code used for the simulation attack is in Appendix A.3. My test to decide about dropping traffic on R is: if the packet came from R or was addressed to R it was allowed to pass. Otherwise the packet was dropped. This was accomplished by comparing the source and destination IP address of the packet with the IP address of the local router. All BGP traffic would fall under those cases because it either originates at the one router, or is addressed to it as BGP only communicates with direct neighbors. The attack is enabled through a configuration parameter in the .dml file. It can be switched on and off on different machines in the simulation. A simple code change would also allow remote control of the attack. Then an attacker could send a signal to the machine from elsewhere to start the attack. In this way multiple misprogrammed routers could all execute the malicious code starting at the same time.

3.3 Possible Damage

The damage such an attack would cause could be tremendous. A careful attacker could place several modified network stacks at routers throughout the Internet. If the

routers are well placed, most traffic could come to a halt. The fact that so little code was changed, and that code was changed within the operating system, the malicious code would likely go unnoticed. Unfortunately the attack is only a symptom of the damage that could have been done. In order for the attack to be launched, the router was compromised. Therefore, to truly protect the system the weakness that allowed the intrusion must be discovered and eradicated. Once the problem is found, the operating system would need to be reinstalled without the malicious code. Then the appropriate security patches need to be applied as well. Such a system overhaul could take a day or more for an unprepared system administrator and ruin or delay the network for that time. If the attack occurred in a widespread area with key routers, the impact would be much greater.

In an attack on an individual machine, the Oblivious Engineer attack would do more damage to the network than a complete shutdown of the machine. The Oblivious Engineer attack could easily be activated once the adversary has full control of the machine. The attack would be preferable to taking down the entire router as BGP can recover from a total link failure if there is another route. It will be frustrating for users who may get a short service interruption and a slower connection but it may not even require human intervention to keep the network running. Instead, in the Oblivious Engineer attack the network will fail with no signals to the administrator of the machine. The only feedback would be from user complaints about the problem. We now focus on the proposed solution to such an attack.

Chapter 4

Extended BGP

To combat the Oblivious Engineer Attack, we have developed a countermeasure called Extended BGP. It is only an extension of BGP because it uses much of the functionality of BGP, leaving the core BGP routine alone. The basic algorithm has several steps. It must recognize failure in the network, probe for new routes and use the new routes. There are many choices of implementation of each step as discussed in the following section. Then we give the proposed solution called Extended BGP, and explain it further using pseudocode. As with any research, the solution had implementation issues which are covered in section 5.4. We discuss what design decisions were made as well as what options are still left open to the client or future implementers.

4.1 Countermeasure Steps

To remedy the Oblivious Engineer Attack, we need a monitor on when traffic is getting through the network. There are several places we could monitor the network. For instance, each router could keep track of all the different connections, but that presents many problems. First, current routers could not keep track of the many thousand connections they would handle because it would require too much additional memory and processing power. A router that could handle such computation would be exceptionally expensive. More importantly, many connections are not symmetric. Often times traffic will follow one path from source to destination, and a different path

on the way back. For instance, if traffic is being sent from AT&T in California to an MCI network in Boston, AT&T might put the traffic on MCI's cross country link on the way to Boston, and MCI would put the traffic for California on AT&T's link because neither wants to carry the traffic on their expensive cross country link. For these two reasons, it is unreasonable to expect every router to monitor its connections for reachability. In fact, it has been expressed in the "end to end" [12] argument that the best design is to put the intelligence of the network in the end nodes and keep the connecting nodes simpler. Therefore, the proposed solution considers how end nodes might monitor the network for such failures. We look to the source of the connection to determine if a problem is serious enough to consider. The source, and only the source, knows if the data needs to be timely, like a web connection, or could wait and try again later, like an email. Therefore we choose the source to determine the importance of the traffic and to notify the BGP speaker, as no one else has that information.

Another question is how alternative routes can be found. Our general approach is to test the network for working routes and use the viable routes. The first step is to test for complete routes. One option is for the client to probe for problems in the network. This may lead to many probes all to the same destination from different clients on different hosts throughout the network. This would create a large amount of duplicated efforts. The same destination may be tested by several clients within an AS all looking to connect. Their results would all be identical because they all used the same route. Bandwidth would also be wasted. We could consider having the client decide on new traffic routes. Unfortunately, the client has little choice in routing decisions and can still do no better than sending it to one of very few routes out of their AS. Often times the number of BGP speakers is small (1-3 total). Therefore the client is not the best choice for probing and choosing routes as it will create redundant efforts and be of limited usefulness.

The entities that do have more information on routes and more control over the traffic destination are the BGP speakers that are at the edge of the enclave (AS). They need not do redundant checks of the same routes for multiple clients. Likely the same

solution discovered a few seconds earlier will also be the solution to another host's connection failure complaint. BGP speakers have more options available to them, as they are connected to more varied networks. We look to combine the strengths of both the client and the BGP speaker by having the client raise the alert about a route when it fails, and the BGP speaker exploring other options as available. In this way, we need only explore a route once for the entire enclave instead of multiple times for multiple sources. In addition, the BGP speaker may have multiple route options that it can evaluate and controls what route is used.

4.2 Proposed Defense Mechanism Overview

In total we propose that each individual client monitor their connections for failure, and determine themselves how important each connection is and whether it needs to be fulfilled so immediately. The client is then responsible for sending to the BGP speaker the address of the machine with which it cannot communicate. The BGP speaker collects this information and explore the path for itself. It will determine if the current path is viable or not by attempting to make a connection to the machine. If the connection cannot be established and the route to the server has failed, the BGP speaker will consider if other routes may work. Finally, based on policy within the speaker, it can choose a new solution or simply decide that no acceptable path is available. Based also on policy, the BGP speaker may respond to the complaints in order of importance of client or based on the frequency of complaints from different hosts.

4.3 Extended BGP Mechanism Details

There are two distinct parts to Extended BGP, the client and the server. The client runs as part of any host that wishes to participate and the server runs on at least one BGP speaker for an AS. The pseudo code will document the mechanism as written and then we will later discuss what the key design decisions were and what other

options may be taken for added benefit.

4.3.1 Extended BGP Client

The extended BGP client is a very simple one. It has the following four steps. We refer to the address of the TCP client as C, and the address of the TCP server as S.

1. Detect the failure when communicating with another machine named S.
2. Connect to the Extended BGP server in local AS.
3. Send S to Extended BGP server.
4. Close connection.

The basic steps are the same for any Extended BGP client. The placement of such code and exact implementation may differ depending on the caller of the code. If an end-host server detects a failure and wants to run the Extended BGP client, calling it can be somewhat tricky. We discuss the difficulties with a server calling for Extended BGP in section 4.4.1. If a client needs to call the Extended BGP client, it is generally very simple.

4.3.2 Extended BGP Server

The Extended BGP server is more complicated than the client as it does almost all of the work for the protocol. It runs constantly on a BGP speaker, waiting for others to make a connection to it. The code is based on the TCP server code. If at any point in the following description a command fails, or an outcome is not mentioned, then it stops and returns. Such outcomes are of no central importance to our task.

```
Initialize the server by:
  -binding to a socket on a well known port
  -listening on that socket
```

```
Accept the socket when a connection is made to it:
  -create a new slave server to service the call
  Slave Server services the call:
    -read from the socket the IP address S
    -close the connection
  Check with the request-manager:
    -Send 'S' to request manager
    -Receive response giving permission to research S
  Ping S:
    -Send multiple packets to S requesting a response on currently
     used route to S
    -Wait 20 seconds for a response
       -If we receive a response, the path is valid and we halt
          the inquiry. Otherwise,
  Remove routing entry for current path to S from forwarding table:
    -Read which neighbor advertised S from the forwarding table
    -Remove entry for S from local list of that neighbor's
     advertised paths
    -Run normal BGP decision process code to reselect new best route
      -If no alternate route can be found, restore old route
      -BGP will automatically advertise the change to its neighbors
```

4.4 Implementation Issues

The implementation of extended BGP was often challenging but generally straightforward. The code of SSFNet was well organized, very modular, and mostly well commented. Therefore when adding in the additional functionality and attack, we were careful to maintain the modularity. During the process, we made several design decisions that were key to the outcome of the service discussed in section 4.3.2. There were many other decisions that have been left up to future work and the imple-

menter that we also consider in section 4.3.3. First though, one difficulty arose in our implementation that was not solved to our satisfaction. Although the code works, it is a hacked solution that did not seamlessly fit into pre-existing architecture as gracefully as desired. It required modification of the general socket API (Application Programming Interface).

4.4.1 Socket API Modification

One less than ideal situation arose when implementing the Extended BGP client as part of a TCP server. When the server listens for a new connection and then accepts it, it creates a new socket. This new socket is put into the incompleted sockets queue waiting for the ACKs (acknowledgements of receipt) back from the client. In this case, they will never come because the connection is broken. Normally, the socket silently times out and is removed from the uncompleted socket queue; the accept function would not return an error. This is a problem because we would like to detect such failures and consider if we should have Extended BGP investigate the problem. We changed the semantics of the accept function such that it returns an error when an incomplete socket times out. Then the server is aware of the failure and can choose to complain to the Extended BGP server. However to send a complaint to the Extended BGP server, we need to know the IP address of the client that could not be reached. This information is not returned by accept and so we created a new function to retrieve that IP address. The function, called last_request, returns the source IP address from the most recent timed out incomplete socket. Alternatively, we could have changed the semantics of accept to automatically complain to the Extended BGP server. However, we chose to have accept return an error to allow the caller of the socket API to make the decision to call Extended BGP or not. In total, the code does work, but adding to the socket API is less modular and involves changing standard API code. At this point in time we saw this as the best choice of implementation. Future work would hopefully produce a more elegant solution.

4.4.2 Design Decisions

In developing Extended BGP we made several key design decisions. We needed to decide where the code would reside on the machine. Should it be part of the network stack, the operating system, or a normal process? We also looked at how the Extended BGP client and server should communicate. Another area where there were several options was to look at how to test the route to determine if it is working.

Early on, we considered where in the system the code for Extended BGP should exist. It would seem that the best place for the extended BGP code to be is within the operating system as it is a behind the scenes mechanism which needs a great deal of privilege. On the client side, the code need not be part of the system as it is so simple, but could be part of an application. The top level application would know how important each connection is to make, and if the extra effort should be made for the connection to get through in such a timely manner. Just as easily, such an application could make a system call to initiate Extended BGP as well. The location of the Extended BGP client code has many possibilities. In the current implementation the application makes all of the calls on the Extended BGP client. At the Extended BGP server in my simulation, the code runs as an application but with perhaps more privileges than a normal application on a server would run. In reality it seems that it may be more realistic to have the code run as a system process, always running in the background with liberal permissions. This will enable consistency across different platforms. With both sections of code as part of the operating system, the system calls used to initiate the code are simple and the Extended BGP server has the permissions it requires.

A second key design decision we made was about the protocol used to communicate between the Extended BGP server and client. In our current implementation, the client connects to the server using TCP. Instead we may want to consider using UDP[7] instead of TCP to connect to the server. The server makes no assurances of processing each and every command, therefore, it is also reasonable to use UDP as it similarly makes no promise of packet delivery. UDP uses less bandwidth and

processing and one merely hopes that the commands arrived and were processed. The request could always be made again if the path continues to fail.

We considered the many possible ways to check for response of a server, and in our implementation we used the common ping command which uses ICMP services. This works well as ICMP is widely deployed and used already for network monitoring. Ping is a simple protocol with very minimal overhead using only UDP packets and needing no information beyond what host to contact and how many packets it should send. Other options for Extended BGP to test the route might include trying to make the exact same connection that the end host attempted. This is too specific as it might fail because of many other problems including failure of the specific service it was trying to access. Extended BGP merely looks to make sure that a path is available to the server, and can make no assurances about the correct functioning of that server. Another alternative to ping is to use traceroute to verify the path to the server. Traceroute sends out a series of packets with different short TTLs (Time To Live), then when a packet is purposely dropped at each router along the way, it sends back an error message to the source. In this way the source learns who all the routers are between it and the server it wants to contact. Then the Extended BGP server might learn more about the path and where it is failing. Additional functionality could be built into the Extended BGP server with that information, but we chose instead the simpler method of ping in the attempt to keep the network relatively simple.

Another issue with using ping packets to test a server response against the Oblivious Engineer attack is that attack could be fairly easily modified to also let ping packets pass. Unfortunately, this is the case with any specific protocol used to connect to the server. The best hope is that probing traffic is hidden within the normal traffic profile and does not stand out for easy processing.

4.4.3 Extended BGP Server Options

My implementation is only one of many choices for determining if routes should be explored and changed. When a complaint about a server arrives, one could service

the complaints in many different orders, or not at all. For instance, the complaints could be processed in order of arrival, or they could be processed on order of which host is the most important. Alternatively, they could service the complaint that has been made most often by the entire enclave in total. It could also outright drop a percentage of complaints and process those that remain, solving some of the problems. Most crucial is that the server does not assure processing of any of the complaints. It must be free to forget about some and drop them, otherwise it leaves the server open to a denial of service attack. If if is acceptable to drop some complaints, as UDP may drop packets, then Extended BGP will be much less affected by any such attacks. Unfortunately, it is still open to traditional attacks such as SYN flooding, as currently implemented. This is another reason to use UDP instead of TCP to communicate between the Extended BGP client and server.

It would seem important, in one way or another, not to service requests too quickly and change routes too often. This might lead to route flapping and instability in the network. We might only check complaints about the same router that are at least 10 minutes apart. An initial attempt at moderating requests is implemented in the simulation as the request manager which is consulted before investigating a complaint.

One may also consider different options about when to share your changed routes. Normally, BGP would share this information immediately upon changing routes, but the effect of such aggressive change should be explored further. Extended BGP should also be responsible for continuing to check if previously removed routes are functioning again. Currently, those routes are discarded never to be considered again until the neighbor advertises them. As the route was originally the most preferred one, it should be used if possible. This would also be important in the situation of non-universally deployed Extended BGP. If the next hop does not run Extended BGP and therefore does not withdraw its route because it does not detect the failure, the next hop will not advertise it again for our BGP speaker to re-learn about it. The BGP speaker would be responsible itself for maintaining that information.

One strategy of route switching, which we did not employ, but would seem necessary after running the simulation is to verify that whatever route is switched to

next, will respond to a ping of the server. Perhaps the server might itself be down, and there is no problem with the network. The Extended BGP must not assume too much and make a different conclusion. If all routes are examined and none of them prove to connect to the network, then BGP could remain the same and not pick a different route. Otherwise Extended BGP will find an alternate route that does work, and reconnect the network. It may not find a route that will return a proper response because the return path in all cases also uses the faulty router. It is impossible to differentiate between this case and when all paths are truly faulty. Then switching routes may be an effective solution, as it cannot make the problem any worse.

Chapter 5

Simulation and Results

In this chapter we cover the motivation for performing the simulation, the details of said simulation, and its results. We consider what security risks may be introduced by Extended BGP. We also look at what future work might be done in more elaborate simulations to further evaluate the mechanism. The simulation was performed using SSFNet, an open source network simulation package described further below.

5.1 SSFNet Simulation Program

In order to simulate my ideas within a controlled environment, we used SSFNet. It is an "open-source Java model of protocols, network elements, and assorted support classes for realistic multi-protocol multi-domain Internet modeling and simulation" [1]. It is one of the few simulation programs that implements BGP specifically instead of a generic link-state or distance-vector protocol to simulate routing. The source can be found at http://www.ssfnet.com. Using this program we were able to specify different network layouts, different connections, protocols, and the network timing. It takes input through a .dml file, a Domain Modeling Language file which is explained in section 5.1.2. SSFNet is also easily extendable by adding to the source code. To implement my new protocols and attack modules we simply added to the source code new objects and methods. SSFNet is based on the basic Scalable Simulation

[1] www.ssfnet.com

Framework (SSF) which is a "public-domain standard for discrete-event simulation of large, complex systems in Java and C++." It is "'the engine under the hood' of the SSFNet Internet models." [2]

Another interesting note about the SSFNet simulation is that it is written using continuations to enable simulation of the multi-threaded environment. This was both powerful and frustrating at different points. In some cases, it simplified the code greatly to have both failure and success continuations available to be called at the appropriate moment. At other times, in order to change the timing and wait for certain events before proceeding without blocking other computation, the code was less clean and more convoluted.

5.1.1 BGP Simulation Code

The BGP4 simulation code[10] was mostly contributed by BJ Premore as an extension to SSFNet. It is now released as part of the full current version of SSFNet. The code is written to conform to all applicable standards provided in the RFCs on topic. It does not yet implement Secure BGP.

5.1.2 DML Configuration Files

The configuration files were written in Domain Modeling Language to express the network layout and protocols to be used. It did not require me to give specific IP addresses to hosts or subnetworks. Instead, the SSFNet assigned them. We were able to control the layout of the autonomous systems and control their naming. The basic format is simple. A network is defined using keyword *net*, specify the machines in the network (*host* or *router*), and list what protocols they run. Additional attributes for protocols were simply put into a NAME VALUE list within the protocol definition. An example of my network layout is in Appendix A.1.

[2] www.ssfnet.com/homePage.html

5.1.3 Risks

Using the SSFNet simulation program is somewhat risky in that it is an open-source implementation that is still currently being developed and expanded. In the process of our work we did discover several bugs and report them to the appropriate people. A risk does remain though that other bugs that are not so noticeable may have affected my simulation results.

5.2 Why Run a Simulation?

We wrote the simulation of the Oblivious Engineer Attack and Extended BGP solution for a number of reasons. Primarily we wanted to be sure that the ideas really worked. Writing the code, and working through the other network protocols inspired many new ideas as well as reminders of small details that had not been considered. It also gave us an idea of the difficulty of implementation and what bugs were likely to occur. Finally, we wanted to look at how our protocol would work in connection with others in the network to see if its results were valuable.

5.3 Network Design for Extended BGP Simulation

The network design we chose is a five router network, with two end hosts attached to two different routers. Let C (1:101) be the name of the TCP client and S (2:202) be the name of the TCP server. All of the routers in this example are the BGP speakers for their AS. Only the important nodes are shown, for instance the routers without displayed attached hosts would presumably have such hosts, but they are not key to the simulation. The simplicity of the network gives a reasonable expected behavior from the Extended BGP servers as there are only two paths from C to S. When one path fails it should automatically discover the other.

To test Extended BGP we ran several TCP connections across the network looking to get data from S to C. The router 4:1, as shown in Figure 5-1 implements the Oblivious Engineer attack dropping all traffic other than BGP traffic. When the first

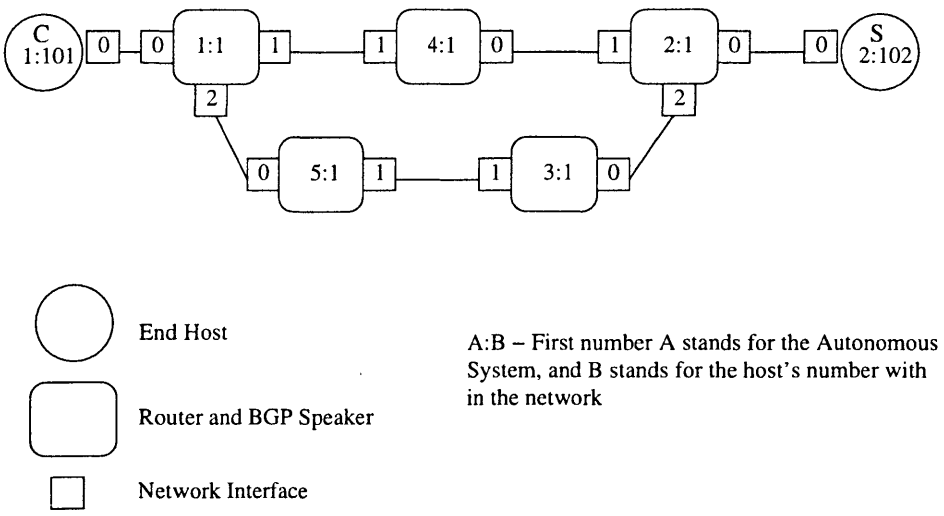


Figure 5-1: Extended BGP Simulation Network

connection fails, Extended BGP discovers the problem and finds the alternate path through which to send data to the end nodes. The exact output of the simulation is shown in Appendix A.2. The explanation of timing of events in plain English is in the following sections.

5.3.1 Simulation Events Timetable

Time (in seconds)	Event Happening
0.0	Simulation Begins
57.5	Regular BGP Stabilizes - all routes known
71.0	C sends connect message to S
71.1-100	Connection traffic dropped by 4:1
145.6	C connection attempt timed out
145.6	C connects to Extended BGP server on 1:1
145.6	Extended BGP server 1:1 reads IP address of S, sent by C
145.6	Extended BGP server 1:1 pings S
145-149	Ping packets dropped by 4:1
155.6	Extended BGP server 1:1 gets no ping replies from S, removes the route, finds a new one and advertises the new route
200.0	C (2nd connection attempt) sends a new TCP request to S
200.1	S receives connection request from C and responds
201 - 230	4:1 drops S's response traffic.
274.6	C times out and sends message to Extended BGP server
274.6	Extended BGP server 1:1 receives complaint, but finds that it has already been investigated and does nothing.
275.0	S times out with no reply from C
275.0	S connects to its local Extended BGP server 2:1 and complains
275.4	Extended BGP server 2:1 pings for C
276-279	Ping traffic is again dropped by 4:1
285.5	Extended BGP server 2:1 gets no ping response, faulty route removed and a new one put in its place
300.0	C (third attempt) connects to S and transmits its data perfectly.

5.4 Results

As shown in the previous timetable, we send three connection requests across the network, with it taking two mangled requests to get the third clear connection working. The first connection allows C to complain about the network to the Extended BGP server 1:1 as its packets never make it across the network. Then the route from C to S is corrected. The second TCP connection, appears to do the same thing to C, but in fact its packets have made it across the network and S has replied but the S's response packets were lost. This is an example of non-symmetrical routing because the server is still using 4:1 to route packets back to the host. It knows no better because it has not yet tried out the route. Once this connection fails at S because it did not get a response, it too will initiate a connection with its local Extended BGP server. Then for the final connection, both routers are using the alternate route and avoiding router 4:1 and so no traffic is dropped along the way.

5.4.1 Overall Traffic Impact

The overall traffic impact of this solution seems quite minimal. The new traffic is the TCP connection between C and the Extended BGP server, and the ping requests from the Extended BGP server and S. A few additional BGP messages may also be sent when the new routes are discovered and used. These additional traffic requests, when compared to the original traffic planned on being sent, are quite small. Therefore the impact on the traffic of the network will be minimal in terms of total volume, although certain parts of the network may experience more congestion because of the need to avoid another route.

5.5 Known Issues with Simulation

The simulation provided valuable insight into how the protocol should work and might work in a real world situation. It brought to light the difficulties in having bi-directional communication that uses different paths. It also provided a different

coding environment than a real router and network stack would be written. We had access to many different variables that may or may not be accessible so easily in a real router. For instance, in a real router the extended BGP code may reside within the network stack, or as a separate process running on the machine to be accessed. It may have different privileges depending on where it is placed as such.

The simulation is merely that, a simulation, not a full real world experience with many different variables interacting. One could never simulate all the possible scenarios that the real world might bring. Other simulations though could provide more insight into how the protocol should be modified to succeed.

5.5.1 A More Ideal Implementation

Ideally we would like the connections to recover in fewer than three communication attempts. One such way to do that might be to skew the client and server time out lengths. For instance if the server were to time out before the client, the second TCP connection attempt in the simulation might be salvaged and reconnected if the server were to send its packets again. While this would produce a valid connection in two attempts instead of three, it is a hack to solve a tricky problem. One cannot control all the timers in the Internet and these timers have widespread impact because of their length and so it would be a poor choice to change them to solve a lesser problem.

Alternatively, we might like to see the first computer act that notices the network has failed between it and another host. It might try to send information to the distant host letting it know that it know about the lack of response. This information is transferred eventually in the current protocol, as the second attempted connection fails. It could be sent earlier if the first node recognizes that the connection has failed and sends a message out to the server with which it wishes to speak. In order to assure a connection if any possible path exists, the client would need to send a packet out all possible routes from its AS. If the server were to receive any of the packets, it could alert its own extended BGP server and have it probe the connection before the second failed attempt is started. In this way we may save a failed connection timeout, but end up sending more packets overall.

Difficulties with this strategy include the added traffic, but also the issue of trust. Why should a distant server trust the information it is getting from a random client. In doing so it may leave itself open to denial of service attacks. While denial of service attacks are very real and a problem, there are easier protocols to misuse to create such an attack, and Extended BPG only adds one more similar path to do the same thing. It's not adding a new weakness, it is only slightly widening an already available attack.

5.6 Impact on Routing Infrastructure

There are millions of machines on the Internet, and each router needs to know the route to get to every one of them. In order to save memory, each router lists similar IP addresses together in their forwarding table if they both use the same path. For instance, 1.1.1.1 through 1.1.1.7 are most likely in the same subnetwork and therefore the path to the AS that contains them both is the same. Therefore the router uses a shorthand notation of 1.1.1.0/3 to say that the first 29 bits of the IP address are the only significant ones. Each IP address has 32 bits in it. The 3 (32-29 = 3) signifies that that if the first 29 bits match 1.1.1.0, then the last 3 bits can be any possible combination and the two addresses should be considered a match. Then its forwarding table will have 1 entry instead of 8 for the same path to the network. When Extended BGP changes a route, we may find routes being updated to only a single host, and so we may separate router listings like the one above. This unfortunately will grow the forwarding tables and may require that the router increase its memory.

Another impact Extended BGP may have on the routing infrastructure involves the amount of state required in the BGP speakers. A BGP speaker needs to keep track of what routes it has changed recently, what servers it has pinged and is awaiting responses from, and what complaints have been made. While this extra state will require more memory, it can be controlled through many measures. First, the Extended BGP server makes no promises of servicing each and every request. Therefore, if the queue of complaints grows too much, it can ignore new complaints until

the list is short enough. Second, if too many outstanding pings have been sent, it is not crucial that every non-responsive server be explored as well. Additional memory will be required to handle this state but the server can control how much memory it uses while keeping Extended BGP functioning properly.

5.7 Using Extended BGP Against the Network

As we introduce the Extended BGP mechanism into the network to provide additional security, we need to consider how Extended BGP might be misused and harm the network. There are three scenarios to consider: an insider only attack, an outsider only attack, and collusion of an insider and an outsider.

An insider only attack might consist of a host sending fraudulent complaints to its Extended BGP server. Then the Extended BGP server would test the routes, find that they work, and do nothing more. This attack has minimal impact as it only requires some processing power of the BGP speaker and adds ping packets to the networks. The Extended BGP speaker could also implement a policy to ignore a host if it makes too many successive complaints and also reduce its workload.

An outsider attack consists of a machine that is in one AS trying to trigger the effects of Extended BGP in a different AS. This could be prevented by using packet filtering at all BGP speakers. The speakers would filter for all packets that enter through a non-local interface addressed for the Extended BGP server on any machine. They would be easy to spot as Extended BGP runs on a single specified port.

A collusion attack, where both a host within the AS and a host outside of the AS cooperate may be much stronger. The insider would instigate a complaint about the route to the outsider. The outsider could then purposefully ignore the attempts to connect to it with pings. Then the Extended BGP server would determine that the route had failed. If the outsider responded to no ping connection attempts, then Extended BGP would not change its preferred route since it believes that no routes are working. It would make no changes to its existing forwarding table, the traffic would all be sent along the same route and delivered if possible. Therefore in order

for the collusion attack to have an impact, the outsider would have to respond to some pings and not others. In this way, the attackers might change which path is used and they have successfully altered the network. While changing routes without reason is not preferable, it is not a grave consequence. An Extended BGP server would not choose a route that is unacceptable to it (i.e. Microsoft would not send its confidential traffic over an IBM network). Therefore, the routes may have been changed, but the traffic will still pass, and all parties will agree to the route, or the Extended BGP server will not change the route.

5.8 Further Simulation

Additional simulations or real world experiments are desirable to thoroughly test the mechanism. The following are two ideas for such experiments that look promising at this point.

5.8.1 Interaction with BGP and Networks on a Larger Scale

A large simulation that would be very enlightening is to look at how Extended BGP might look on a larger scale. It should be placed in an environment where everyone is running Extended BGP. It should also be considered in a mixed environment where only some routers run Extended BGP, such as might be expected if the protocol were gradually adopted. The experiment could also be run on a real set of machines with humans initiating the traffic for more randomized input and realistic results.

5.8.2 Attack with Different Intervals

Another useful simulation would be to look at how an Oblivious Engineer Attack that is stopped and started would confuse the network. Would the attack confuse Extended BGP more or just let the network converge more quickly because it does function part of the time?

Appendix A

Network and Code Details

A.1 Network DML File

This network was the general network used to simulate and test the functions of Extended BGP. A diagram of the network can be found in section 5.3. Lines preceeded by '#' are commented out and have no impact on the network.

```
_schema [ _find .schemas.Net ]
Net [
  frequency 1000000000   # nanosecond simulation resolution

# Options to print out data from BGPs workings
  bgpoptions [
     show_rcv_update true # Show when BGP receives an UPDATE message
  ]

  # AS #1 Network - includes 1 router running Extended BGP and one
  # host that is running the TCP client C. Network is pictured in
  # Figure 5-1.

  Net [
    AS_status boundary
```

```
id 1
ospf_area 0

# BGP Speaker and Extended BGP Server for AS 1. Router 1.
router [
  id 1
  graph [
    ProtocolSession [
      name bgp
      use  SSF.OS.BGP4.BGPSession
      autoconfig true
    ]
    ProtocolSession [
      name exbgpserver
      use ExBGPServer
      debug true
      request_size 4
      port 1600
    ]
    ProtocolSession [ name icmp use SSF.OS.ICMP ]
    ProtocolSession [
      name socket
      use  SSF.OS.Socket.socketMaster
    ]
    ProtocolSession [
      name tcp
      use  SSF.OS.TCP.tcpSessionMaster
    ]
    ProtocolSession [
      name ip
      use  SSF.OS.IP
    ]
```

```
  ]
  # This router has three interfaces and a default route out
  # interface 0.
  interface [ idrange [ from 0 to 2 ] ]
  route [ dest default interface 0 ]
]

# Host C (101) which runs the 3 tcp client attempts to S.
host [
  id 101
  graph [

    ProtocolSession [
      name newtcpclient
      use newTcpClient
      debug true
      start_time 71.0
      file_size 50000
      request_size 4
      localBgpNhi 1:1(0)
      localBgpPort 1600
    ]
    ProtocolSession [
      name newtcpclient_after
      use newTcpClient
      debug true
      start_time 200.0
      file_size 5000
      request_size 4
      localBgpNhi 1:1(0)
      localBgpPort 1600
    ]
```

```
      ProtocolSession [
        name newtcpclient_should_work
        use newTcpClient
        debug true
        start_time 300.0
        file_size 5000
        request_size 4
        localBgpNhi 1:1(0)
        localBgpPort 1600
      ]
      ProtocolSession [ name icmp use SSF.OS.ICMP ]
      ProtocolSession [
        name socket use SSF.OS.Socket.socketMaster ]
      ProtocolSession [ name tcp use SSF.OS.TCP.tcpSessionMaster
        tcpinit [
          MaxRexmitTimes 5
          show_report true
        ]
      ]
      ProtocolSession [ name ip use SSF.OS.IP ]
    ]
    interface [ id 0 ]
    route [ dest default interface 0]
  ]
  # attach router 1 interface 0 to host 101 interface 0
  link [ attach 1(0) attach 101(0) delay 0.001 ]
]

#  Autonomous System Number 2. Contains one router (1) and one
#  host (102). The host runs the TCP server S that C attempts
#  to connect to. The router is the BGP speaker and also an
#  Extended BGP server.
```

```
Net [
  AS_status boundary
  id 2
  ospf_area 0

  # Router 2:1 which runs BGP, Extended BGP and ICMP
  router [
    id 1
    graph [
      ProtocolSession [
        name bgp
        use  SSF.OS.BGP4.BGPSession
        autoconfig true
      ]
      ProtocolSession [
        name exbgpserver_server
        use ExBGPServer
        debug true
        request_size 4
        port 1600
      ]
      ProtocolSession [ name icmp use SSF.OS.ICMP ]
      ProtocolSession [
        name socket
        use  SSF.OS.Socket.socketMaster
      ]
      ProtocolSession [
        name tcp
        use  SSF.OS.TCP.tcpSessionMaster
      ]
      ProtocolSession [
        name ip
```

```
      use  SSF.OS.IP
    ]
  ]
  interface [ idrange [ from 0 to 2 ] ]
  route [ dest default interface 0 ]
]

# Host S which runs the TCP server.
host [
  id 102
  graph [

    ProtocolSession [
      name server
      use newTcpServer
      port 2000
      client_limit 10
      request_size 4
      show_report true
      debug true
      localBgpNhi 2:1(0)
      localBgpPort 1600
    ]
    ProtocolSession [ name icmp use SSF.OS.ICMP ]
    ProtocolSession [ name socket use SSF.OS.Socket.socketMaster
      debug true
      strict_accept true
    ]
    ProtocolSession [ name tcp use SSF.OS.TCP.tcpSessionMaster ]
    ProtocolSession [
      name ip
      use  SSF.OS.IP
```

```
      ]
    ]
    interface [ id 0 bitrate 5000 ]
    route [ dest default interface 0]
  ]
  # attach router to router
  link [ attach 1(0) attach 102(0) delay 0.1 ]
]

# Two identical BGP Speakers for different autonomous
# systems (3 or 5). No hosts shown to simplify the simulation.
Net [
  AS_status boundary
  id (3 or 5)
  ospf_area 0

  router [
    id 1
    graph [
      ProtocolSession [
        name bgp
        use  SSF.OS.BGP4.BGPSession
        autoconfig true
      ]
      ProtocolSession [ name icmp use SSF.OS.ICMP ]
      ProtocolSession [
        name socket
        use  SSF.OS.Socket.socketMaster
      ]
      ProtocolSession [
        name tcp
        use  SSF.OS.TCP.tcpSessionMaster
```

```
      ]
      ProtocolSession [
        name ip
        use  SSF.OS.IP
      ]
    ]
    interface [ idrange [ from 0 to 1 ] ]
    route [ dest default interface 0 ]
  ]
]

# Autonomous System 4 which contains the evil IP stack that
# implements the Oblivious Engineer Attack. No other hosts
# or routers.
Net [
  AS_status boundary
  id 4
  ospf_area 0

  router [
    id 1
    graph [

      ProtocolSession [
        name bgp
        use  SSF.OS.BGP4.BGPSession
        autoconfig true
      ]
      ProtocolSession [ name icmp use SSF.OS.ICMP ]
      ProtocolSession [
        name socket
        use  SSF.OS.Socket.socketMaster
```

```
      ]
      ProtocolSession [
        name tcp
        use  SSF.OS.TCP.tcpSessionMaster
      ]
      ProtocolSession [
        name ip
        use  Attacked_IP
          # Where the evil comes from.
        evil true
      ]
    ]
    interface [ idrange [ from 0 to 1 ] ]
    route [ dest default interface 0 ]
  ]
]

# Attach the interfaces of the routers together so that
# they are connected as in Figure 5-1.
link [ attach 1:1(2) attach 5:1(0) delay 0.01 ]
link [ attach 4:1(1) attach 1:1(1) delay 0.01 ]
link [ attach 5:1(1) attach 3:1(1) delay 0.01 ]
link [ attach 4:1(0) attach 2:1(1) delay 0.01 ]
link [ attach 3:1(0) attach 2:1(2) delay 0.01 ]

# Controls the traffic pattern within the network dictating
# that any client on 1:101 is to speak with the server on
# 2:102 through port 2000.
traffic [
  pattern [
    client 1:101
    servers [nhi 2:102(0) port 2000]
```

```
    ]
  ]
]
```

A.2 Simulation Output Results

```
## Net config: 7 routers and hosts
## Elapsed time: 1.47 seconds
** Running for 1000000000000 clock ticks (== 1000.0 seconds sim time)
------------------------------------------------------------------------
| Raceway SSF 1.0b09 (2 May 2001)
|  Proprietary Internal Development Release
| (c)2000,2001 Renesys Corporation
------------------------------------------------------------------------
# Initialize the randomness in each client so they are not syncronized.

0.0 newtcpclient_should_work 1:101 0.0.0.10
      MersenneTwister::DefaultStream/{Scheduler 0}
0.0 newtcpclient_after 1:101 0.0.0.10
      MersenneTwister::DefaultStream/{Scheduler 0}
0.0 newtcpclient 1:101 0.0.0.10
      MersenneTwister::DefaultStream/{Scheduler 0}

# Regular BGP is distributing routes
5.081115852   bgp@3:1     rcv update frm bgp@5:1 nlri=0.0.0.44/30,asp=3
5.081115852   bgp@3:1     rcv update frm bgp@2:1 nlri=0.0.0.0/29,asp=2
5.081115852   bgp@4:1     rcv update frm bgp@2:1 nlri=0.0.0.0/29,asp=2
5.081115852   bgp@4:1     rcv update frm bgp@1:1 nlri=0.0.0.8/29,asp=1
5.081115852   bgp@5:1     rcv update frm bgp@3:1 nlri=0.0.0.36/30,asp=5
5.081115852   bgp@5:1     rcv update frm bgp@1:1 nlri=0.0.0.8/29,asp=1
5.081115852   bgp@2:1     rcv update frm bgp@4:1 nlri=0.0.0.40/30,asp=4
```

```
5.081115852   bgp@2:1     rcv update frm bgp@3:1 nlri=0.0.0.36/30,asp=5
5.081115852   bgp@1:1     rcv update frm bgp@5:1 nlri=0.0.0.44/30,asp=3
5.081115852   bgp@1:1     rcv update frm bgp@4:1 nlri=0.0.0.40/30,asp=4
27.766900764  bgp@1:1     rcv update frm bgp@4:1 nlri=0.0.0.0/29,asp=4 2
27.766900764  bgp@2:1     rcv update frm bgp@4:1 nlri=0.0.0.8/29,asp=4 1
29.396697042  bgp@1:1     rcv update frm bgp@5:1 nlri=0.0.0.36/30,asp=3 5
29.396697042  bgp@3:1     rcv update frm bgp@5:1 nlri=0.0.0.8/29,asp=3 1
30.352841166  bgp@5:1     rcv update frm bgp@1:1 nlri=0.0.0.40/30,asp=1 4
30.352842692  bgp@4:1     rcv update frm bgp@1:1 nlri=0.0.0.36/30,asp=1 3 5
30.373140347  bgp@4:1     rcv update frm bgp@1:1 nlri=0.0.0.44/30,asp=1 3
30.373140347  bgp@5:1     rcv update frm bgp@1:1 nlri=0.0.0.0/29,asp=1 4 2
31.261414319  bgp@4:1     rcv update frm bgp@2:1 nlri=0.0.0.36/30,asp=2 5
31.261415845  bgp@3:1     rcv update frm bgp@2:1 nlri=0.0.0.8/29,asp=2 4 1
31.271566198  bgp@2:1     rcv update frm bgp@4:1 wds=0.0.0.36/30
31.2817135    bgp@3:1     rcv update frm bgp@2:1 nlri=0.0.0.40/30,asp=2 4
32.018037172  bgp@2:1     rcv update frm bgp@3:1 nlri=0.0.0.8/29,asp=5 3 1
32.018037172  bgp@5:1     rcv update frm bgp@3:1 nlri=0.0.0.40/30,asp=5 2 4
32.038334827  bgp@2:1     rcv update frm bgp@3:1 nlri=0.0.0.44/30,asp=5 3
32.038334827  bgp@5:1     rcv update frm bgp@3:1 nlri=0.0.0.0/29,asp=5 2
32.048486706  bgp@3:1     rcv update frm bgp@5:1 wds=0.0.0.0/29
50.462816193  bgp@1:1     rcv update frm bgp@4:1 nlri=0.0.0.36/30,asp=4 2 5
50.462816193  bgp@2:1     rcv update frm bgp@4:1 nlri=0.0.0.44/30,asp=4 1 3
53.722408749  bgp@1:1     rcv update frm bgp@5:1 nlri=0.0.0.0/29,asp=3 5 2
53.722408749  bgp@3:1     rcv update frm bgp@5:1 nlri=0.0.0.40/30,asp=3 1 4
57.451843303  bgp@4:1     rcv update frm bgp@2:1 nlri=0.0.0.44/30,asp=2 5 3

# First connection attempt from C to S.
71.0 newtcpclient 1:101 0.0.0.10 sid 1 connect()
71.0 TCP host 1:101 src={0.0.0.10:10001} dest={0.0.0.2:2000}  Active Open

# Traffic Dropped by Evil router 4:1
71.011261034 host 4:1 in ip: Attacked_IP Dropped Traffic From: 0.0.0.10
```

```
76.574383295 host 4:1 in ip: Attacked_IP Dropped Traffic From: 0.0.0.10
100.574383295 host 4:1 in ip: Attacked_IP Dropped Traffic From: 0.0.0.10

# First Connection attempt times out, and C connects to the Extended
# BGP Server on 1:1
145.563122261 newtcpclient 1:101 0.0.0.10 ETIMEDOUT
145.563122261 newtcpclient 1:101 0.0.0.10 sid 1 connect()
145.563122261 TCP host 1:101 src={0.0.0.10:10002} dest={0.0.0.9:1600}
                Active Open
145.566547381 exBGPtcpServer 1:1 0.0.0.9:1600 slave read(), request check
                of 0.0.0.2 OK
145.566547381 exBGPtcpServer 1:1 0.0.0.9:1600 serviceRequest() OK

# Extended BGP Server on 1:1 pings S
# Connections are closed between S and the Extended BGP server
145.566547381 exBGPtcpServer 1:1 0.0.0.9:1600 Pinging this ip: 0.0.0.2
145.567677898 newtcpclient 1:101 0.0.0.10 sid 1 complaint about 2:102(0) OK
145.567677898 TCP host 1:101 src={0.0.0.10:10002} dest={0.0.0.9:1600}
                Active Close
145.567677898 TCP host 1:101 src={0.0.0.10:10002} dest={0.0.0.9:1600}
                Passive Close
145.567677898 newtcpclient 1:101 0.0.0.10 Sucess on calling the
                ExBGP Client
145.569969449 newtcpclient 1:101 0.0.0.10 sid 1 close() OK

# Evil Router 4:1 drops ping traffic from Extended BGP Server
145.576705364 host 4:1 in ip: Attacked_IP Dropped Traffic From: 0.0.0.30
146.576705364 host 4:1 in ip: Attacked_IP Dropped Traffic From: 0.0.0.30
147.576705364 host 4:1 in ip: Attacked_IP Dropped Traffic From: 0.0.0.30
148.576705364 host 4:1 in ip: Attacked_IP Dropped Traffic From: 0.0.0.30
149.576705364 host 4:1 in ip: Attacked_IP Dropped Traffic From: 0.0.0.30
```

```
# Extended BGP Server ping request times out, finds new route
# removes the old one and advertises the change.
155.566547381 exBGPtcpServer 1:1 0.0.0.9:1600 No ping response received,
                must modify BGP tables
0.0.0.2/32 <- ip to find
Route found in table:{dst=0.0.0.0/29 :nxt=0.0.0.29 :cost=-1 :adist=20
                :if=[NIC 1:1(1) 0.0.0.30/30 :: bitrate=1.048576E7,
                latency=1.0E-4, buffer= 9223372036854775807] :src=EBGP}
Peer found!
155.566547381 exBGPtcpServer 1:1 0.0.0.9:1600 Extended BGP removed a route
155.57669926  bgp@5:1     rcv update frm bgp@1:1 wds=0.0.0.0/29
155.576717571 bgp@4:1     rcv update frm bgp@1:1 nlri=0.0.0.0/29,asp=1 3 5 2

# Second connection attempt made from C to S
200.0 newtcpclient_after 1:101 0.0.0.10 sid 1 connect()
200.0 TCP host 1:101 src={0.0.0.10:10003} dest={0.0.0.2:2000}  Active Open

# TCP Server on S receives the request and creates an incomplete socket.
200.131652585 tcpSocket host 2:102 port 2000 - add incomplete socket,
                incompl_queue= 1

# S's traffic to C is dropped
200.305883101 host 4:1 in ip: Attacked_IP Dropped Traffic From: 0.0.0.2
206.156758882 host 4:1 in ip: Attacked_IP Dropped Traffic From: 0.0.0.2
230.156758882 host 4:1 in ip: Attacked_IP Dropped Traffic From: 0.0.0.2

# C's 2nd connection attempt times out and C contacts Extended BGP on 1:1
274.563122261 newtcpclient_after 1:101 0.0.0.10 ETIMEDOUT
274.563122261 newtcpclient_after 1:101 0.0.0.10 sid 1 connect()
274.563122261 TCP host 1:101 src={0.0.0.10:10004} dest={0.0.0.9:1600}
                Active Open
274.566547381 exBGPtcpServer 1:1 0.0.0.9:1600 slave read(), request check
```

```
                of 0.0.0.2 OK
274.566547381 exBGPtcpServer 1:1 0.0.0.9:1600 serviceRequest() OK

# Extended BGP gets the request, but sees that the route has already
# been checked and altered. So it ignores the request.
274.566547381 exBGPtcpServer 1:1 0.0.0.9:1600 0.0.0.2 has been previously
                complained about. Ignoring request.
274.567677898 newtcpclient_after 1:101 0.0.0.10 sid 1 complaint about
                2:102(0) OK
274.567677898 TCP host 1:101 src={0.0.0.10:10004} dest={0.0.0.9:1600}
                Active Close
274.567677898 TCP host 1:101 src={0.0.0.10:10004} dest={0.0.0.9:1600}
                Passive Close
274.567677898 newtcpclient_after 1:101 0.0.0.10 Sucess on calling the
                ExBGP Client
274.569969449 newtcpclient_after 1:101 0.0.0.10 sid 1 close() OK

# S times out, removing the incomplete socket from the queue and
# accept returns with a failure
274.982528366 tcpSocket host 2:102 port 2000 - failure: ETIMEDOUT
274.982528366 tcpSocket host 2:102 port 2000 - removed from incomplete
                queue, queue size= 0
274.982528366 tcpSocket host 2:102 port 2000 - Strict Accept is returning
                upon error - failure: ETIMEDOUT
274.982528366 tcpServer 2:102 0.0.0.2:2000 accept() FAILURE: ETIMEDOUT
274.982528366 tcpServer 2:102 0.0.0.2:2000 About to complain to BGP.
274.982528366 tcpServer 2:102 0.0.0.2:2000 0.0.0.10 Address of where we
                want to test the path

# S connects to Extended BGP server 2:1 and complains
274.982528366 server 2:102 0.0.0.2 sid 0 connect()
274.982528366 tcpSocket host 2:102 port 0 - connect to 0.0.0.1:1600
```

```
275.246758882 tcpSocket host 2:102 port 10001 - write object 4B
275.481258881 exBGPtcpServer 2:1 0.0.0.1:1600 slave read(), request
                check of 0.0.0.10 OK
275.481258881 exBGPtcpServer 2:1 0.0.0.1:1600 serviceRequest() OK

# Extended BGP on 2:1 pings for C and 4:1 drops the traffic.
275.481258881 exBGPtcpServer 2:1 0.0.0.1:1600 Pinging this ip: 0.0.0.10
275.491416864 host 4:1 in ip: Attacked_IP Dropped Traffic From: 0.0.0.22
275.581389398 server 2:102 0.0.0.2 sid 0 complaint about 0.0.0.10 OK
275.581389398 tcpServer 2:102 0.0.0.2:2000 Sucess on calling the
                ExBGP Client
275.909619913 server 2:102 0.0.0.2 sid 0 close() OK
276.491416864 host 4:1 in ip: Attacked_IP Dropped Traffic From: 0.0.0.22
277.491416864 host 4:1 in ip: Attacked_IP Dropped Traffic From: 0.0.0.22
278.491416864 host 4:1 in ip: Attacked_IP Dropped Traffic From: 0.0.0.22
279.491416864 host 4:1 in ip: Attacked_IP Dropped Traffic From: 0.0.0.22

# Extended BGP server 2:1 gets no response from C and therefore
# modifies the forwarding tables to use a different route.
285.481258881 exBGPtcpServer 2:1 0.0.0.1:1600 No ping response received,
                must modify BGP tables
0.0.0.10/32 <- ip to find
Route found in table:{dst=0.0.0.8/29 :nxt=0.0.0.21 :cost=-1 :adist=20
                :if=[NIC 2:1(1) 0.0.0.22/30 :: bitrate=1.048576E7,
                latency=1.0E-4, buffer= 9223372036854775807] :src=EBGP}
Peer found!
285.481258881 exBGPtcpServer 2:1 0.0.0.1:1600 Extended BGP removed a route

# New updates are sent to neighbors to reflect the new route.
285.49141076  bgp@3:1    rcv update frm bgp@2:1 wds=0.0.0.8/29
285.491429071 bgp@4:1    rcv update frm bgp@2:1 nlri=0.0.0.8/29,asp=2 5 3 1
```

```
# Third TCP connection attempt from C to S
# which works as all traffic passes through!
300.0 newtcpclient_should_work 1:101 0.0.0.10 sid 1 connect()
300.0 TCP host 1:101 src={0.0.0.10:10005} dest={0.0.0.2:2000}  Active Open
300.131652585 tcpSocket host 2:102 port 2000 - add incomplete socket,
                incompl_queue= 1
300.458927237 tcpSocket host 2:102 port 2000 - read object 4B
300.458973014 tcpServer 2:102 0.0.0.2:2000 slave read(), request 5000B OK
300.458973014 tcpSocket host 2:102 port 2000 - write data 5000B
302.296120081 newtcpclient_should_work 1:101 0.0.0.10 sid 1 request 5000B
                to 2:102(0) OK
314.020553862 [ sid 1 start 300.0 ] newtcpclient_should_work 1:101 srv
                2:102(0) rcvd 5000B at 2.852kbps - read() SUCCESS

# All open connections slowly close down
314.020553862 TCP host 1:101 src={0.0.0.10:10005} dest={0.0.0.2:2000}
                Active Close
314.152206447 tcpServer 2:102 0.0.0.2:2000 serviceRequest() OK
314.543481098 tcpServer 2:102 0.0.0.2:2000 slave close() OK
385.509135523 exBGPtcpServer 1:1 0.0.0.9:1600 slave close() OK
515.009135523 exBGPtcpServer 1:1 0.0.0.9:1600 slave close() OK
515.735419213 exBGPtcpServer 2:1 0.0.0.1:1600 slave close() OK
554.063122261 TCP host 1:101 src={0.0.0.10:10005} dest={0.0.0.2:2000}
                2MSL timeout, connection closed
554.063122261 newtcpclient_should_work 1:101 0.0.0.10 sid 1 close() OK
-------------------------------------------------------------------
| 1 timelines, 5 barriers, 13872 events, 2869 ms, 6 Kevt/s
-------------------------------------------------------------------
```

A.3 The Altered Evil Network Stack

This is the code used as the IP part of the network stack for evil router 4:1. It is almost entirely identical. It differs only when IP must decide if the packet should be sent to this machine, or forwarded on to another.

```
/* Attacked_IP.java */

import java.util.*;
import com.renesys.raceway.DML.*;
import SSF.Net.*;
import SSF.Net.Util.*;
import com.renesys.raceway.SSF.*;
import SSF.OS.*;

/**
  * This class should behave in the following manner described. Instead it
  * looks at each packet and if the packet does not originate at this
  * machine or is not destined for it, it is dropped on the floor if the
  * appropriate option (evil) is set to true.
  *
  * This class implements a subset of the IP protocol. The primary job
  * of IP is to find a route for each datagram and send it on its way.
  * In order to allow gateways or other intermediate systems to forward the
  * datagram, it adds its own header (IpHeader.java).  This version
  * of IP uses a 'next hop' IP address as a surrogate hardware address;
  * future alternatives with more elaborate link layer models will fix
  * this.
  * <pre>
  * Revisions ato 9/24/00: support for configuration of Monitors.
  * Revisions jhc 1/29/01: support for ICMP messages before drop()
  * </pre>
  * @see SSF.OS.IpHeader
```

```
 */
public class Attacked_IP extends IP {

    public boolean evil = false;
    public int delay_evil = 70;
    public int evil_duration = 100;

//------------------------------------------------------ CONSTRUCTOR

  public Attacked_IP() {
    super();
  }

//------------------------------------------------------ CONFIGURATION

  public void config(Configuration cfg) throws configException {
    super.config(cfg);
    String str;
    // Evil - If the protocol should drop all non-local packets or not.
    str = (String)cfg.findSingle("evil");
    if (str != null)
evil = Boolean.valueOf(str).booleanValue();
  }

// ------------------------------------------------------ Push
// Code that handles a message being pushed up the network stack.

  public boolean push(ProtocolMessage message, ProtocolSession fromSession)
              throws ProtocolException {
    IpHeader ipHeader = (IpHeader) message;

    // 1. If the local host is the destination indicated by the ipHeader,
```

```
    //    push the payload up to the correct protocol.

    boolean isLocal = // loopback address 127.0.0.1 with netmask 255.0.0.0
(0x7F000000 == (0xFF000000 & ipHeader.DEST_IP));
    boolean isFromLocal =  false;

    // Test if the packet is destined for this host
    for (int n=0; !isLocal && n<INTERFACE_COUNT; n++)
      isLocal = (INTERFACE_SET[n].ipAddr == ipHeader.DEST_IP);

    // Attacked IP specific code
    if (evil)
        {
         // Check if packet was produced locally
         for (int n=0; !isFromLocal && n<INTERFACE_COUNT; n++)
           isFromLocal = (INTERFACE_SET[n].ipAddr == ipHeader.SOURCE_IP);

           if (!isFromLocal && !isLocal)
                {
                    drop(ipHeader);
                    System.err.println(debugIdentifier() +
                    "Attacked_IP Dropped Traffic From: " +
                    IP_s.IPtoString(ipHeader.SOURCE_IP));
                    return false;
                }
        }
    return super.push(message, fromSession);
  } // end of push()
}

/*=                                                                    =*/
/*=  Copyright (c) 1997--2000  SSF Research Network                    =*/
```

```
/*=                                                                    =*/
/*=  SSFNet is open source software, distributed under the GNU General  =*/
/*=  Public License.  See the file COPYING in the 'doc' subdirectory of =*/
/*=  the SSFNet distribution, or http://www.fsf.org/copyleft/gpl.html   =*/
/*=                                                                    =*/
```

Bibliography

[1] BBN Technologies A Verizon Company. http://www.ir.bbn.com/projects/s-bgp/.

[2] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[3] Kevin J. Houle and George M. Weaver. Trends in denial of service attack technology. Technical report, CERT Coordination Center, October 2001.

[4] S. Kent and R. Atkinson. RFC 2401: Security architecture for the Internet Protocol, November 1998. Obsoletes RFC1825. Status: PROPOSED STANDARD.

[5] K. Perine. http://www.cnn.com/2000/tech/computing/05/12/net.amendment.idg/.

[6] R. Perlman. *Network Layer Protocols with Byzantine Robustness.* PhD thesis, MIT, Department of Electrical Engineering and Computer Science, 1988.

[7] J. Postel. RFC 768: User datagram protocol, August 1980. Status: STANDARD. See also STD0006.

[8] J. Postel. RFC 791: Internet Protocol, September 1981. Obsoletes RFC0760. See also STD0005 . Status: STANDARD.

[9] J. Postel. RFC 793: Transmission Control Protocol, September 1981.

[10] Brian J. Premore. http://www.cs.dartmouth.edu/~beej/bgp, 2002.

[11] Y. Rekhter and T. Li. RFC 1771: A Border Gateway Protocol 4 (BGP-4), March 1995.

[12] Jerome H. Saltzer, David P. Reed, and David D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984.

[13] Andrew S. Tanenbaum. *Computer Networks.* Prentice-Hall, second edition, 1996.