Technique to prevent passive snooping/hijacking on the IRC internet protocol

Karan Chadha and Kevin Dawkins

***Abstract*—This paper analyzes a common type of attack on the internet known as snooping. We take that concept and apply it to a real world scenario, IRC. The IRCs infrastructure of server nodes each with their own clients is similar to the structure of the internet allowing the authors to develop a practical solution to this problem drawing a parallel to the main structure of the internet.**

I. Introduction

A. Structure of Content

This paper will present an overview of the IRC protocol. It will relate current work to the work presented in this paper. Next, the paper will present the work done, with an experimental results. Finally, the paper will suggest further work and provide a conclusion.

B. What is IRC?

The IRC (Internet Relay Chat) protocol is a text based conferencing protocol, which has been developed in 1989. The IRC protocol is based on the client/server model, and it is designed to run in a distributed manner. Internet Relay Chat, is a means of talking with anyone else on the internet via your computer keyboard and screen. You can either talk to them directly, person-to person in a private conversation not visible to others through a query (private message) or DCC chat (Direct Client-to-Client chat), or you can join a channel to talk. A channel is like a public chat room with a lot of people, where you can see all of their conversations on your screen and join in yourself if you want. The size of the channel can vary and in fact sometimes you can even create private channels. To access these channels and use IRC, an individual, through a program called a ”client”, connects to a network of IRC servers, machines that allow users to connect to IRC. There are literally hundreds of networks across the globe and thousands of channels to choose from. [1]

C. IRC Network Structure

The simplest architecture consist of a server with multiple clients connect to it. The server will handle message delivery and multiplexing. There are two types of clients: 1) user clients; and 2) service clients. The user clients are text-based interfaces that interactively communicate using IRC. The service clients are used to provide services to user clients, such as providing statistics. In IRC multiple servers form the IRC network. Servers relay all the communications between the clients. All messages from any server are broadcast to all the other connected servers. All servers on a network share and have access to the same information. Each server knows who is on the network, which channels the users are in, and which servers the users are on.

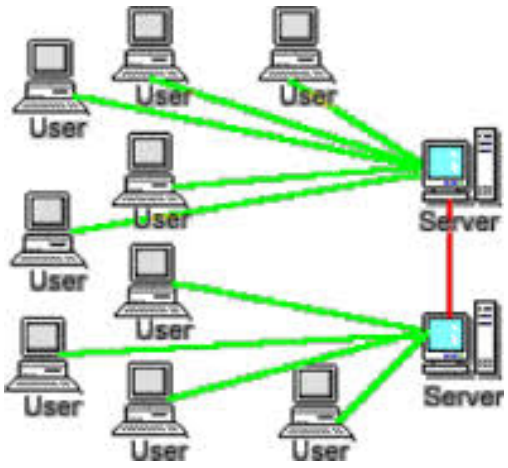

Fig. 1. Sample IRC Network [2]

IRC protocol provide the following services:

- **Client locator**: When a client connects to a server, the client subscribes using a unique label. The server keeps track of all the labels, so that clients and servers in the IRC network can locate the clients and communicate with them.
- **Message Relaying**: Nothing more than message switching, since IRC clients cant directly communicate with one another.
- **Channel hosting and management**: A channel is a multicast communication mechanism among the IRC clients. Each channel is identified by a unique name. User clients can join the channel.

IRC provide the following communication models:

- **One-to-one**: example: a client communicates with another client.
- **One-to-many**: example: a client write to a channel.
- **One-to-all (broadcast)**: example: like a client change of state information.

D. Prefix Hijacking

The internet is divided into Autonomous Systems (ASs) which control large portions of the logical breakdown of the internet. These systems rely on a protocol of announcements to allow other AS system to know what is inside of that specific AS. This improves the reachability of any given

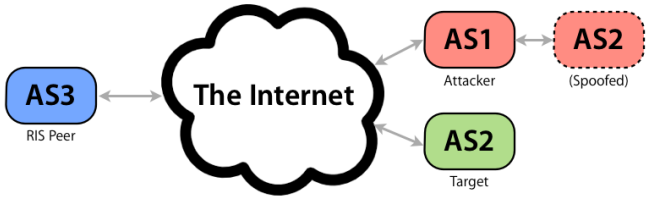

Fig. 2. Snooping Sample [5]

node inside of a single AS. However, recently, attackers (or even sometimes unintentionally), have announced bogus routes either re-routing traffic towards them (snooping), or creating a ”black hole” in the internet. A ”black hole” means that no information can escape because a false route is trapping it. [3], [4]

This is becoming a huge problem since the internet is growing so rapidly, and has a lack of centralized control (by design). The internet is quickly becoming an easy target for hackers and is prone to configuration mistakes that can take down huge parts of it. [3], [4]

E. Snooping

Snooping, in a security context, is unauthorized access to another person’s or company’s data. The practice is similar to eavesdropping but is not necessarily limited to gaining access to data during its transmission. Snooping can include casual observance of an e-mail that appears on another’s computer screen or watching what someone else is typing. More sophisticated snooping uses software programs to remotely monitor activity on a computer or network device.

Related to snooping, we tried to implement prefix hijacking in which our bot tries to steal someone else traffic by getting it routed towards itself. If it is successful the bot can later analyze the traffic, manipulate it and may either redirect it to somewhere else.

In the figure 2, the identity of AS2 gets spoofed by the attacker AS1. Now whenever the two peers AS3 and AS2 will try to communicate their conversation will be sniffed by the attacker. And all this will happen without any knowledge of any server or client attached to the existing network.

II. Related Work

A lot of work has been done in this space both employing cryptography and attempting to solve the problem without the need for cryptography. Specifically, some work has been done trying to deduce bogus routes from the links when compared to the internet as a whole [3]. That paper tried to evaluate new links against the topology of the entire internet to see if a portion started to behave differently. If it noticed that part of the internet stop receiving packets or went ”dark” then it would alert about the bogus connection [3]. Also, the technique described utilizes the other notes to detect when false announcements have been sent. When a node receives a new AS announcement, it compares it with other AS announcements on a local table, then checks if it is consistent.

Other work has been done focusing on how to recover after a prefix has been hijacked. In some cases, an attempt is made to advertise a more specific prefix than the hijacked prefix. However, if an AS filters too long prefixes then this recover method will not work. In that second case, the victim can advertise the same prefix until the attacker stops its attack. Usually both methods are used in a two attempt strategy to recover the victim. [6]

Another interesting paper analyzed the various layers of the internet that an attack of this nature can target. The internet is divided into three different layers (hierarchies): core layer, forwarding layer, and the marginal layer. Each plays a different role on the internet. The paper learned that attacks at different levels can cause varying amount of damage. The paper noted that attacks on the marginal layer took out the most nodes, and had the most problematic effects. However, attacks on the forwarding layer could cause chaos when trying to determine routes. An attack on this layer can cause data redirection which is dangerous for sensitive data. Additionally, the paper determined that while a single attack on the marginal layer affected the most nodes, a coordinated attack on the forwarding layer would have the largest general affect. Finally, an attack on the core layer would be disastrous on the internet. This amount of fragility shows the need for better detection and prevention of attacks.[4]

A very influential paper to this project stressed the need for fast detection of BGP prefix hijacking. Current methods for detecting a prefix hijacking attack are resulting in false alarms, false positives (and false negatives), and slow responses, this is not good as our society moves more onto the internet. Sensitive data cannot wait for an attack to be detected, every second can cost people a lot in terms of privacy and money. The paper makes an interesting observation that played a roll in influencing our project: ”polluted routers usually cannot get a reply from the victim prefix” [7]. After an anomaly in the network is detected, the system deploys the ”Eye of Arugs” which can determine if the anomaly is a false alarm, or a real attack all within seconds. It begins pinging the victim router looking for a reply. [7]

This style was extremely influential in our project. The observation that the attacker makes the victim silent, lead the authors to the observation that some attackers are silent themselves.

[7]

All of these prior work techniques provided incite for the work done on this paper.

III. Proposed Solution

IRC felt like an appropriate medium to conduct research because the characteristics of IRC and attacks on IRC mirror the attacks done on the internet. IRC itself acts like its own internet, turning usernames into clients and passing along messages. This environment allowed the authors to experiment and explore in a real world setting.

A. Implementation

1) IRC Server: The IRC server was implemented in Python and deployed on Windows Azure. It was written in an event driven, scalable style allowing [0-n] clients and [0-m] servers concurrently connected. The server itself supports a subset of the IRC commands, specifically the commands related to joining and creating channels and private messaging. The server was implemented following strict adherence to the IRC protocol. The server did favor the server-server connection when searching for recipients of a the private message command.

One of the challenges on implementing the server was designing it around efficient scale out for channels. As the number of channels and connected users grew, the server had to efficiently handle messages for each channel and compute which users were in each channel. This was solved by storing some state information about each channel meaning more work upfront (when a user connects to a channel) but when the messages started flowing, the server could efficiently handle them.

2) IRC Client: The IRC client was implemented in Perl. Again, it was written in an event driven fashion providing GUI experience with the help of tk library. The client had the ability to open [0-n] channels at the same time. It had the ability to log the channel conversations and later reload them whenever the user joins the same channel. The client was also implemented following strict adherence to the IRC protocol.

3) Hack Bot: The hack bot was also written in Perl. It was a very simple implementation of hacking, where it spoofs the identity of a legit client, the server is searching for. After spoofing it successfully sniffs the messages meant for the original client without informing the server or any other client connected on the network.

When the hack bot first runs, it initializes itself as a legitimate server connection, presenting the correct credentials to the main IRC server (in our case, it passes the 004 code followed by the $server$ phrase). Once this is complete, the hack bot starts listening on the server group. Whenever a private message is sent and the target is not known (perhaps this is the first private message sent to a client), the hack bot announces that it knows the target. Finally, the hack bot intercepts the message then allows the message to be passed on to the actual client; it never acknowledges the receipt of the message so the server keeps searching.

B. Problem and Solution

The IRC implementation is extremely susceptible to the spoofing attack common on todays networks. Though many of the current implementations of IRC attempt to alleviate this problem, the specification leaves a lot of holes. Particularly against an attack where the attacker sits passively on the network and starts announcing that it has paths to some of the other clients on the network. This is especially susceptible to the servers who favor the server server connection. One observation that the authors made about this style of attack is that the malicious server remains passive most of the time. Obviously, this is not the most sophisticated style of attack, but it provides an interesting angle to try to defend against.

These styles of attack only have the malicious server providing some administrative information, initial connection information, and the announcements. Besides those connections, nothing else is contributed to the network. That leaves a big opportunity for the partner-connected server to notice the malicious one. Recall that in IRC, a new server connects to only one server already on the network.

To solve this problem, some design decisions were made from the beginning:

- The solution should be invisible to the joining server.
  - The joining server and clients should not have to provide verification, that can be burdensome for large servers.
- A server that is flagged as malicious should be able to redeem itself.
  - Some IRC servers flood new clients with welcome message and clients, they may look malicious since the new connections do not have a chance to respond.
- The solution should scale and be distributed.
  - This will help prevent single point of failure, and waiting on central servers.

These design decisions do present trade-offs on the implementation of the detection system. However, these trade-offs seem acceptable to the general functionality that the system provides. The trade-offs will be discussed in the *future work* section of the paper.

Enter the *server contribution score*. This number is assigned to newly connected servers and represents their initial level of activity contributing to the network as a whole. The score also is related to two threshold levels. The upper bound represents how much work the new server must do before the network trusts it as an acceptable server. The lower bound represents when the point at which the IRC network will start to suspect malicious activity from the new server, and start to ignore it. As messages begin flowing in and out of that link, the score starts to change. Whenever the new server contributes messages into the network (outgoing) the score rises, and when messages flow into the network (incoming) the score decreases. Even if the server gets a large initial amount of incoming connections (eg. announcements), the score will normalize as the members of that server begin contributing. Administrative announcements (connection info, pings) do not count towards the score either way.

If the server falls below the lower threshold, then its connection gets ignored, but not dropped. The server can certainly redeem itself. As messages start flowing out, the score will renormalize, and the connected network will return to using it as normal.

The opposite happens when the server goes above the upper bound threshold. At this point, the servers in the network can feel comfortable with accepting the server with no suspicion. The score is dropped and the server can receive and send messages without any penalty or score. This shows that the

server contributed a considerable amount to the network and should be trusted.

This is effective against the passive server because their lack of activity will quickly degrade the score and it will be ignored from the network, restoring private peer to peer communication to the affected clients.

Data: t_l : Lower bound
Data: t_u : Upper bound
Data: $curr\_SCS$: target SCS
if $curr\_SCS >= t_u$ **then**
  Accept Server;
else if $curr\_SCS < t_l$ **then**
  Ignore Server;
else
  if *Incoming Connection* **then**
    $curr\_SCS$ = $curr\_SCS + 1$
  else
    Outgoing connection; $curr\_SCS$ = $curr\_SCS - 1$
  end

Algorithm 1: Pseudo Code for SCS algorithm. This code would be run on the server accepting the connection from the new server. It store the SCS data.

IV. Evaluation

A bot was created to help experiment on this network. The bot was able to join the IRC network, introduce itself as a server, and report that it had connections to all clients that the network was searching for. It then read the private messages and printed them to the terminal. Before the *server contribution score* was implemented, the bot was successful and able to both announce it had paths to clients, and intercept transmissions to them.

The *server contribution score* (known as SCS) was successful in helping the main server identify a malicious attacker and ignore them. The bot connected, and initially was able to receive messages until the point of the threshold when the routes became considered invalid (or spoiled), and the server searched for alternative routes. While the server did send a few initial messages to the malicious server before it was identified, this seems acceptable for a few reasons. First, it is hard to judge on connection if the server is malicious or has malicious intent. If one were to assume all incoming connections were malicious, it would have to confirm to the contrary. This could bring in a level of inconvenience for users, making them prove identity, or prove they are human each time the server reconnects to a network. The SCS is a passive system for identifying malicious servers that does not require user intervention.

SCS also reacted well in situations common on IRC networks, a large amount of announcements or floods of information entering the newly connected server. Due to its flexible nature, the server was able to reconnect and start sending messages once again. The score normalized.

V. Future Work

A. Encryption

The need for Encryption/ Decryption arises whenever there is a need for data communication to be protected from others. During network communication whenever a client wants to send the message to another client, in between any attacker can trap the data when it is transmitted through wires or wireless. [8]

On the same note, our solution is based, which is basically just an idea since we never implemented that in the code. The key Idea was that whenever a client would attach any server, it would be provided a private key for decrypting the messages. The server already would maintain the meta data about every client and thus it would know which client has the respective private key. The server would decrypt the messages using the public key which would be meant only for the particular client which has the private key to decrypt it. In this way any other client wont be able to decrypt the messages which is not meant for it. The worst case would be that the attacker could still get the messages but it wont be able to decrypt the messages. [8]

B. Honey Pot

As we have seen in previous attacks, nothing that goes across the network is safe. If the attacker can intercept certain portions of the conversations or session data then he can use that data to impersonate one of the parties involved in the communication so that he may access more private information. Such kind of attacks may also be called as Man in the Middle attack or may even go under the category of session hijacking. These attacks are difficult to detect and even more difficult to defend against because its a mostly passive attack. Unless the malicious user performs some type of obvious action when he accesses the session being hijacked, you may never know that they were there. But the idea in this paper will prevent from such kind of attacks with the help of a honey pot. The key idea is to capture the attacker even before he launches his attack or steals the conversation. [9]

Using the inspiration from the paper and thinking to create the same scenario in our IRC network implementation, we can create a separate server which will have less number of clients. It will act like a low cost/access point without any encryption or authentication. If there is an attacker who would be looking for any such loop holes to snoop into the network, such that the attacker would be able to attach to it easily and our server would act like a honey pot. In this way a hacker activity can be detected and we can save our main network from such dubious actions.

C. Instant Detection

While SCS proved to be very effective against the passive ”black hole” style of attack, it leaves a lot of room for further investigation. Firstly, there are a certain level of messages that get redirected before the system recognizes the bogus link. This can be a problem in sensitive spaces where any level of message redirection can cause problems. However, as discussed above, the authors wanted the system to be

completely passive, meaning that the new server did not need to do any work upfront to prove its validity.

While one of the prior work discussions spoke of a way to detect an attack instantly, it relied on the notion that the victim would be silent due to the takeover by the attacking AS. However, if the attacking AS was snooping and still passing along the data (as in our case), this may not work entirely well.

D. False Negative Identification

While the SCS system is good at identifying false positives, it has gaps when identifying false negatives. For example, any intelligence written into the hacker's tools could periodically send back predetermined (or with natural language processing, computed) responses to the data it receives. If these responses are sent back quickly enough, the SCS system will accept the incorporated server as one of its own, allowing it to carry on with its attack unchallenged. This can be a very dangerous situation. One way of solving this could be analyzing the data being sent back, or allowing feedback from the clients to see if the data is out of the ordinary.

E. Reconnection loop problem

Another area that leaves room for improvement stems from the following scenario:

- Malicious server *A* joins the server group
- *A* begins snooping
- *A* triggers the lower bound of the *server contribution score*
- *A* gets ignored from the server group, and detects the lack of messages
- *A* leaves the group, and reconnects to the same (or a different server)
- *A* begins snooping again
- **Repeat**

This potentially dangerous scenario shows how a server could use a rejoin as a way to get around the *server contribution score*. The server group does not make any attempt to remember which server joined, and even if individual servers did record histories, the malicious attacker could rejoin from another server. The *Server Contribution Score* should be able to announce to the group when a malicious server is detected, and warn the pack. Of course, if the malicious server normalized its score and re-entered the group, it would announce the false alarm.

VI. Conclusion

This project set out to identify a style of attack on the internet. This attack is known as hijacking and snooping, and it relies on fraudulent malicious AS servers sending out bogus paths that may lead nowhere or to the server of a malicious attacker. Specifically, this paper focused on the style of attack consisting of a passive attacker. The attacker only sets up administrative information then passively reads messages as they come in.

The project proposed a solution regarding the introduction of a score assigned to a newly connected server. This score reflected the current state of that server exposing servers that did not contribute to the network as a whole. Those servers were then ignored unless they began contributing to the server group. This effectively cut out the attacker and restored normal service to the IRC group. It also protected against accidental identification of a malicious attacker.

VII. Authors Comments

The entire experience was very enriching. It was rewarding to be able to use IRC as a simulation of certain aspects of the internet rather than a simulator because it provided a level of real world experience. Also, it was interesting to be able to build a client and server taking into account levels of usability, and other factors that real server designers need to consider.

Programming a real server and client model to simulate a network was indeed both challenging and rewarding. SCS also proved to be very successful against the class of attack it was trying to guard against. While this style of attack may not be the most sophisticated, it is a very common style since it is easy to implement and deploy.

SCS also proved to be very flexible because it allows for anomalies to show up on the network, and does not terminate connection under any circumstance. SCS allows for for servers mistaken as malicious to redeem and rejoin the group.

References

[1] J. Oikarinen, "Request for comments (rfc) 1459: Internet relay chat protocol," 1993.

[2] (2013, December). [Online]. Available: http://www.angelfire.com/ca3/luminara/Project/network.html

[3] X. Zhao, D. Pei, L. Wang, D. Massey, A. Mankin, S. Wu, and L. Zhang, "Detection of invalid routing announcement in the internet," in *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, 2002, pp. 59–68.

[4] J. Zhao and Y. Wen, "Analysis on the effect of prefix hijacking attack and internet hierarchy," in *Computer and Information Technology (CIT), 2012 IEEE 12th International Conference on*, 2012, pp. 375–382.

[5] (2013, December). [Online]. Available: https://labs.ripe.net/Members/David_Murray/content-bgp-route-origin-validation

[6] S. Seto, N. Tateishi, M. Nishio, and H. Seshake, "Detecting and recovering prefix hijacking using multi-agent inter-as diagnostic system," in *Network Operations and Management Symposium (NOMS), 2010 IEEE*, 2010, pp. 882–885.

[7] Y. Xiang, Z. Wang, X. Yin, and J. Wu, "Argus: An accurate and agile system to detecting ip prefix hijacking," in *Network Protocols (ICNP), 2011 19th IEEE International Conference on*, 2011, pp. 43–48.

[8] T. VASINYA.S., DEVI PRIYA.B, "Encryption and decryption for secure communication in wlan," S.R.M. ENGINEERING COLLEGE, Tech. Rep., 2005.

[9] (2013, December). [Online]. Available: http://www.excitingip.com/1125/honeypot-man-in-the-middle-attack-wireless-intrusion-prevention/