Dealing with Public Ethernet Jacks – Switches, Gateways, and Authentication

Robert Beck – University of Alberta

ABSTRACT

This paper describes the tools and techniques developed and deployed to address the problem of blocking unauthorized users on unprotected Ethernet jacks. Our solution is being deployed to control public labs at the University of Alberta during the summer of 1999. In this environment, we have a mix of "walk up" Ethernet connections used for laptop computers, and public Windows 95 and 98 workstations with fixed Ethernet connections. By themselves, none of these provide adequate facilities for preventing unauthorized Internet usage and enabling us to track Internet abuses originating from these networks. Prior to the deployment of our new access control system, these networks were not routed off of our campus due to these problems.

Our access control system consists of MAC-locked switches behind a gateway at which an IP filter only allows Internet access when authenticated. Now we allow the authenticated users full access to the Internet, while preventing unauthorized people from plugging in for free Internet access. This also provides a record of Internet activity by authenticated users so that abuses can be easily tracked.

We also have several transparent proxies on the gateway machine to assist us in handling particularly troublesome security and configuration issues relating to the internal lab. This allows us to selectively proxy out bound IMAP, SMTP, and HTTP requests, as well as answering IDENT requests coming in to the lab with the real user. The solution is inexpensive and easy to deploy, using off-the-shelf switches and a gateway router running a free operating system and software.

The Problem of Public Labs and Unsecure Ethernet Jacks

Public labs and their use by students in a university environment have always raised several issues for network managers and those concerned about security. The first goal of the labs is always to allow students to learn, and offer them every amount of freedom possible to do so. At the same time, there is great potential for abuse where public lab hosts can be used to attack other sites. If there are no access restrictions and no monitoring, people off the street can walk in and obtain free Internet access. University students can also usually be counted on to take advantage of a chance to cause mischief. This is especially problematic if it is unlikely that they can be held accountable for their actions.

Before the days of laptops, this problem had been addressed in our environment by allowing unrestricted net access only via hosts running multiuser operating systems such as Unix, assigning logins, and using a variety of tools to enable the behavior of the users to be monitored to some degree. This also ensured that only someone in possession of a legitimate login id and password could use the facilities. In this traditional model, system administrators maintained administrative control of the hosts, and ran an Operating System (OS) allowing them to limit and monitor the activities of the users. Access to the Internet was controlled by access to the machine. With the more common use of laptops and the requirement to run unsecure PC desktop operating systems like Windows 95/98 in public places, this solution was no longer feasible. A new solution was needed that preserved the freedom of users to access the Internet, but maintained the ability to associate specific network traffic with specific users, if necessary. This solution also had to accommodate users plugging their own computers into the network.

The Requirements for a Solution

We decided early on that a solution for us would need to have the following properties:

- It must be easy and inexpensive to deploy
- It must work with a default configuration of Windows 95 preferably with no additional software required, and as little user education as possible.
- It should work with our existing authentication methods on campus (Unix and AFS/Kerberos [3, 7, 10]) – We don't want to hand out another 50,000 login accounts.
- It should not allow unauthenticated users to use the Internet
- It should be as unrestrictive as possible to the users once they are authenticated.
- It should give us the ability to track the user id behind Internet abuse at least as well as we can

track the same for the users of our multiuser UNIX hosts.
- It should work both for locations with unsecure Ethernet jacks and for public PC labs.

Other sites have approached a similar problems by having students register MAC addresses through some mechanism before the user can receive an IP address via DHCP [11], or by the use of commercial firewall software [12, 13, 14].

The commercial firewall solution used by many locations was examined and found wanting for several reasons. The foremost reason was cost – every product we examined was prohibitively expensive for us. Secondarily, the solutions we looked at required either one of two things:

- The user had to do something (usually a telnet and authenticate) to actively "log in" to the firewall to authenticate from an IP address then had to do something else (telnet and authenticate again) to actively "log out". This was a problem for us both from a user training point of view, and from a security point of view (session spoofing prevention is difficult).
- If the above issue was addressed, it was usually by means of a custom modification to the client host's operating system. We could not support these on the scale that our campus would require.

The DHCP MAC registration techniques used by other sites were also considered. Again, we had some difficulties with implementing this at our site, Specifically:

- We now would have to register and support the MAC addresses for all our users, in addition to our existing support for 50,000 campus-wide login ID's – this was a big support issue to us.
- MAC addresses are broadcast for DHCP requests, even on a switch – they are hardly secrets – an attacker can simply configure their computer to use another user's MAC address which they have seen before on a broadcast thereby effectively becoming that user, with us being none the wiser unless the attacker and the victim are attempting to use the system at the same time.
- Our policy, for better or worse, tells our users that they are responsible for keeping their password from anyone else or suffer the potential consequences. We felt it would be unreasonable to add their Ethernet MAC address to the list of secrets that a user was responsible for protecting.
- This solution does not work for labs of PC workstations, so we would need to deploy a different solution there.

Once we decided these solutions did not fit our needs we then looked at implementing our own solution.

Our Solution

Our solution uses the following key pieces:

- An authenticating gateway router. One of these sits in front of each lab.
- A switched Ethernet network consisting of cabling and Ethernet switches.
- Our central Kerberos servers for user authentication.

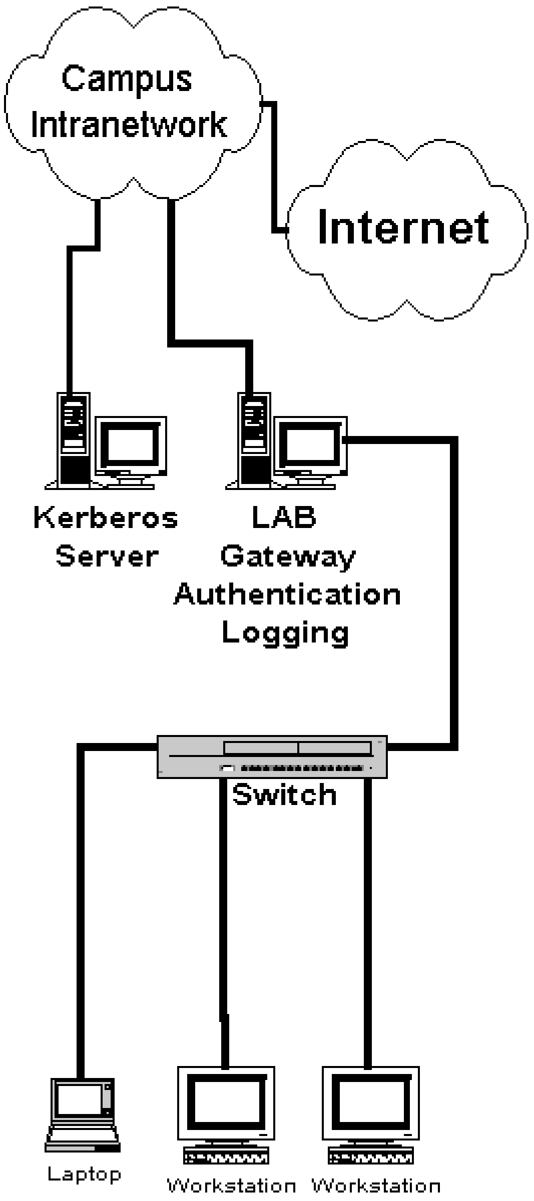

Figure 1: Authenticating gateway.

Figure 1 shows the typical setup in our environment, with the authenticating gateway located at the perimeter of the Lab network. The authenticating gateway router is configured using packet filters to by default block all traffic from the lab network. The switches in the lab allow the user to connect to the gateway router. A user gains Internet access by opening a telnet connection to the gateway, and then authenticating with a Kerberos id and password. Successful authentication automatically creates packet filter rules that permits the user to route through to the rest of the campus and to the Internet, as long as the telnet connection remains open. When the authenticating telnet connection is closed, the filters are removed, so Internet access from the connecting address is no longer allowed. The gateway logs all the TCP connections to the Internet, as well as the login/logouts of each user.

Switches

All our public labs have been configured to use a switched Ethernet network. Properly configured, these will normally ensure that traffic will remain separated, and that a user on one port of the switch will not be able to see or gain control of traffic to or from another port of the switch. This is a requirement for our solution, as we use a standard telnet connection for the authentication of the user.[1] Without this our users would be vulnerable to session snooping and hijacking of the telnet connection used for authentication.

In order for the switches to provide the needed level of security, the switches need to have certain features and be set up correctly to use them. For the labs consisting only of PC workstations, the configuration of the switch is relatively simple. We set our switches to enable *sticky learned* port security, effectively tying each port to one Ethernet address. The port is then disabled if the Ethernet address is changed [1].

For the unsecure Ethernet Jacks used with a laptop, this doesn't work. We need to have the ability for an Ethernet address on a port to change, so the switch must be set to learn new addresses as someone plugs a new machine into a port. This creates a problem. Most switches (including ours) will by default flood unknown unicast packets to all ports. This means that if an attacker can make a legitimate user's Ethernet address unknown (usually by sending many new addresses down a port for a switch to learn) he will then see traffic for the legitimate user as it is broadcast out all ports. This can cause problems with our use of an un-encrypted telnet session. To prevent this we disable unicast flooding on the switches. [1] With unicast flooding disabled, unknown unicast packets are dropped by the switch, changing this sort of attack from a potential compromise of user passwords and authentication to a merely annoying denial of service.

[1]We could use a secure connection, such as ssh, but PC OS's don't come with such software.

An Authenticating Gateway

For the gateway we had to implement software to authenticate users and manage the packet filters. We chose to do this on OpenBSD [15] as it is free, has all the relevant pieces we need in the OS, an emphasis on source code audited for security, and free source code that we can modify to suit our needs. We configure the ipf packet filtering facilities in OpenBSD to, by default, block all traffic originating from the lab side of the network. We then wrote a small program and modifications to /usr/bin/login so that the users could telnet to the gateway and authenticate to get Internet access. To facilitate the client setup, we use an interface alias on the internal interface of all our gateways to answer to a private [6] IP address – 10.0.0.1. This lets us configure any client to telnet to the same place for authentication, and they will always connect to the correct gateway machine for their current location on our network.

By default, our packet filter rules on the gateway are set up similar to the following in /etc/ipf.rules; see Listing 1. When the user authenticates, a program authipf is run which does the following:

1. Log the fact that the user has connected and authenticated.
2. Add filter rules to allow all traffic from the connecting IP address, logging all TCP connections.
3. Wait for the connection to go away
4. Removed the filter rules from above
5. Log the fact that the user has disconnected, and the duration of the session.

```
## allow the inside to talk to ME, to telnet for authentication
pass in quick on fxp0 from any to 129.128.38.65/32

# We use an interface alias on the internal address, so all clients
# can be configured to telnet to one address and they aren't
# different in each lab.
pass in quick on fxp0 from any to 10.0.0.1/32

# log all inbound TCP connection requests
pass in log quick on ep0 proto tcp from any to 129.128.38.64/26 flags S/SA

# otherwise deny anything in on internal interface from the internal net
# that does not get allowed by subsequent (added by authipf) rules.
block in on fxp0 from any to any
```

Listing 1: Packet filtering rules.

authipf adds two filter rules similar to:

```
pass in log quick proto tcp
    from 129.128.38.100/32
    to any flags S/SA
pass in quick
    from 129.128.38.100/32
    to any
```

authipf removes the same rules when it exits. These rules pass all IP traffic, logging the TCP connections.

For authentication, our campus already makes use of Kerberos [3, 7]. We have an existing user base of approximately 50,000 accounts, one for every student and staff member. We did not, however, want to manage accounts on the lab gateways, since users there did not require login access to the machine, only the ability to authenticate. We modified OpenBSD's login program slightly for our purposes. The changes were very simple:

- If the user exists locally, proceed with the normal login procedures.
- Otherwise, if the user does not exist locally, try the username/password in Kerberos. If this succeeds, run authipf for the user/IP address. Otherwise, fail.

With these pieces in place, we have everything we need to control and log the out bound Internet access from the lab, ensuring only authorized users are able to access the Internet. Legitimate users simply open a telnet connection to the gateway and authenticate. As long as the user leaves open the telnet connection to the gateway, the gateway allows their traffic to pass. As soon as the user is disconnected from the gateway, traffic is no longer allowed to pass. The user needs nothing special beyond a telnet client on their machine. Since unauthenticated traffic is not allowed out, any attempts by the users to send out packets with a bogus source address are blocked by the lab gateway. This means that we do not need to worry about this sort of activity putting put a load on our permiter routers.

A user authenticating, using the net, and then exiting might look something like Listing 2.

In this case, we see a user (beck) authenticating from a lab host (129.128.38.100), at 10:42 AM, and then disconnecting 67 seconds later, with some telnet and web activity in between time. Note that by default, we do not log non-TCP traffic (UDP, ICMP) explicitly. In practice this has not proven to be a limitation, as most abuses (i.e., ping floods, udp mischief) are easily tracked by the times they occur.

Problems and Solutions

One problem we encounter with Ethernet jacks in a public place is that of users unplugging their laptop from the net and walking away without closing the session. In this case, we need the gateway to know that the user has gone away, and to remove filter rules allowing their machine access to the Internet. We tuned the TCP KEEPALIVE values on the gateway machine to ensure that sessions would be expired promptly should the user unplug a laptop and walk away. We decided that for our purposes it was adequate for connections of this type to go away in under a minute.

Another possible problem is a user injecting bogus ARP replies into the network to take over an IP address. This is a problem, since our authentication of a user is based on the IP address they are using. OpenBSD itself watches and notices in the syslog when ARP values change. While this is a regular occurrence in a plug-in environment, it can also be used as an attack. To handle this, we can use Swatch [2] to watch the logs for the ARP entry being overwritten for a particular IP address, and then ensure that any running authipf process for a particular IP address is killed, so the IP address is not allowed to pass to the Internet until it authenticates again. While this creates a possible denial-of-service attack within the lab, it does not allow a user to inject bogus ARP replies into the network and take over an authenticated IP address to gain Internet access. As soon as the gateway detects the change in the ARP table entry, any old authentication info and filters for that IP address are removed.

Another risk to keep in mind is the fact that in this environment we use switches configured with

```
Aug  1 10:42:03 law104-gw authipf[27571]: Allowing 129.128.38.100/32,
                                                          user beck
Aug  1 10:42:14 law104-gw ipmon[17403]: 10:42:13.826803 fxp0 @0:27 p
129.128.38.100,1049 -> 129.128.125.13,23 PR tcp len 20 48 -S
Aug  1 10:42:14 law104-gw ipmon[17403]: 10:42:13.826803 fxp0 @0:27 p
129.128.38.100,1049 -> 129.128.125.13,23 PR tcp len 20 48 -S
Aug  1 10:42:02 law104-gw ipmon[17403]: 10:17:51.737302 fxp0 @0:25 p
129.128.38.100,1064 -> 216.33.151.7,80 PR tcp len 20 48 -S
Aug  1 10:43:02 law104-gw ipmon[17403]: 10:30:11.730452 fxp0 @0:25 p
129.128.38.100,1062 -> 216.33.151.7,80 PR tcp len 20 48 -S
Aug  1 10:43:10 law104-gw authipf[27571]: Removed 129.128.38.100/32,
                                         user beck - duration 67 seconds
```

Listing 2: Authenticating, using the net, and exiting.

some unusual options. The security of this system depends on the switch being configured to either have sticky learned port security turned on, or unicast flooding disabled. This could be easily "overlooked" by people performing maintenance on the switch. Our campus uses SNMP monitoring systems as well as regular review by paranoid individuals to ensure that this risk is minimized to an acceptable level for us. One must also ensure that the switch is configured not to accept any sort of management connection from the internal network, lest someone attempt to bypass security by breaking in to the switch and changing the switch configuration.

Further Support for the Lab Environment

With one gateway machine and switches, we now have a "one-box" solution to our public laptop needs. We simply run dhcpd on our OpenBSD gateway to provide DHCP [5] service to laptop clients, in addition to acting as the authenticating gateway.

In addition to merely authenticating the users, and allow/disallowing connections based on packet filtering techniques, with a gateway machine in place we also have the ability to intercept and transparently proxy certain requests. We do this in several situations.

The gateway machine proxies inbound IDENT [8] requests, and answers them on behalf of the internal clients. We run IDENT servers on most of our centrally administered hosts so other on-campus hosts may query the IDENT server to learn the identity of a remote user. On our gateways, we capture IDENT requests for the client addresses and we then answer them with the username used to authenticate out, enabling remote system administrators to query for and obtain user names using IDENT when they receive connections from our labs.

For some of our locations, we gain some additional security and ease of client configuration by storing the user and password information on the gateway and using this in transparent proxies. On the gateway we are able to catch and proxy IMAP [4] protocol connections to our central IMAP server. We have a simple proxy which watches for the IMAP LOGIN command, and replaces the arguments to the command with the user's real login ID and password, as used to authenticate from the connecting address. Connections to other IMAP servers are not captured, and so pass through unchanged. We also intercept out-bound SMTP [9] access to our central SMTP server, replacing the sending address with the user's real e-mail address (derived from the login they used to authenticate). As with the IMAP case, connections to other SMTP servers are not captured and will pass unchanged. While the ability to do this has some interesting security implications, we do it mainly to allow easy set up of an e-mail client in the PC labs, so that the configurations of the client don't need to change from user to user.

Problems This Doesn't Solve

Many people have been, and continue to be, concerned that this does not address the issue of "What if the user gets up and walks away." The filter rules are not tied to any sort of activity monitoring to detect an active or idle connection and timeout, although authipf has an optional overall timeout value. We chose not to implement an activity based timeout for several reasons:

- We don't think users will be likely to leave their laptops and walk away, so it's only a problem when using labs of PC's.
- It doesn't solve the problem. If the user walks away someone can simply pick up where they left off before the timeout value.
- It's annoying to the users to constantly have to re-authenticate if they are periodically accessing the Internet for information, while doing their own work which does not involve Internet access.
- We already are well versed in dealing with the problem of users leaving themselves logged in on the regular timesharing systems – we deal with this effectively already as a "People Problem" that doesn't require a technical solution.

This does not address several denial-of-service issues or problems of intra-lab abuse (from one workstation to another). These are addressed by more traditional (technical and non-technical) means in our labs.

It is important to remember that this is a method for providing authenticated Internet access. It does little (or nothing) for the security of the individual client stations. (If the bad guy is already on the user's laptop or PC, this doesn't help much).

Deployment and Use

Our initial test deployments of this system were deployed in front of networks in our student residences, and in front of one large scale laptop Ethernet jack deployment in September 1998. It has served us very well with very few problems. As of Summer 1999 we are deploying this system in front of all of our public PC labs and public Etherenet jacks. Important to know is we typically do not actively monitor what people are doing in the labs looking for problems. If we don't receive complaints, then we don't really care what students are doing. On the other hand, when we do receive a complaint about something originating from one of these locations, finding the offending user has been a very simple process of looking in the logs for the evidence at the right times. If we had wanted to restrict wholesale what users were able to do, we would have taken a different approach, likely something more akin to a traditional firewall.

Currently we are the process of deploying this system in front of 35 lab locations (20-30 workstations each) throughout our campus, as well as a number of other laptop plug locations. The deployed gateways have disk adequate to store log records for over six months, which is as much as we feel we will need for our purposes.

Conclusions

We have so far been very happy with the level of control this system has given us when deployed in front of our unsecure public places. It is entirely based on free software, works well in our fully switched environments. It integrates well with our Kerberos authentication (and could be easily made to work with others). It allows the users full network access, and does not require any custom software on the client hosts at all. User training has been relatively painless, consisting of a poster at the front of the lab, as well as an icon added to the standard desktops of the workstations. The system completely prevents unauthenticated users from using public Ethernet jacks to gain Internet access. The logging produced as a bonus is thorough and usable to track abuse by authenticated users very effectively when a complaint is received. With the addition of the IDENT proxy, many complaints have already been resolved without even resorting to the logs ("Yes, you can believe the IDENT – thanks"). If you are looking for a solution to dealing with public Ethernet jacks and Internet access, this is a great solution that scales very well, at very little cost.

Availability

Our code is available under BSD-style license terms, and can be obtained from ftp://sunsite.ualberta.ca/pub/Local/People/beck/authipf/ Our gateways were all built using OpenBSD, Much of the code used to construct this system was taken from OpenBSD and modified to suit our needs. See http://www.openbsd.org for OpenBSD.

Author Information

Bob Beck has a Masters degree in Computing Science from the University of Alberta. He has worked in a variety of systems administration and programming positions at the University of Alberta since 1990. He also works as a consultant, instructor, and programmer with Obtuse Systems Corporation. He is currently the Secure Systems Specialist for the University of Alberta, as well as working on several free software projects. You can reach him by postal mail at Computing and Network Services; 352 General Services Building, U of A Campus, Edmonton, Alberta, Canada, T6G 2H1. You can reach him by e-mail to beck@bofh.ucs.ualberta.ca, or beck@obtuse.com.

References

[1] *Catalyst 1900 Series Installation and Configuration Guide* Part Number 78-4362-01, 1997, Cisco Systems Inc.

[2] Stephen E. Hansen, E. Todd Atkins, *Automated System Monitoring and Notification with Swatch*, USENIX Association's Proceedings of the Seventh Systems Administration (LISA VII) Conference (1993) p. 145-155.

[3] J. Steiner, C. Neuman, and J. Schiller, *Kerberos: An Authentication Service for Open Network Systems*, Usenix Association's Conference Proceedings, Dallas, Texas, February, 1988, p. 191-202.

[4] M. Crispin, *RFC2060: Internet Message Access Protocol – Version 4rev1*, December 1996.

[5] R. Droms, *RFC2131: Dynamic Host Configuration Protocol*, March 1997.

[6] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear *Address Allocation for Private Internets*, February 1996.

[7] J. Kohl, and C. Neuman *The Kerberos Network Authentication Service (V5)*, September 1993.

[8] M. St. Johns, *RFC1431: Identification Protocol*, February 1993.

[9] J. Postel, *RFC821: Simple Mail Transfer Protocol*, August 1982.

[10] AFS filesystem information from Transarc Corporation, http://www.transarc.com/Product/EFS/AFS/index.html .

[11] ResNet at University of California, Davis, http://resnet.ucdavis.edu/ .

[12] ResNet at University of Auburn, http://webserv.duc.auburn.edu/hotline/Contents/Resnet/ .

[13] ResNet at Buffalo, http://wings.buffalo.edu/computing/documentation/win/CampusAccess/firewall.html .

[14] ResNet at Wright State, http://www.cats.wright.edu/catsweb/residence/ .

[15] The OpenBSD project, http://www.openbsd.org/ .