OpenBSD/x-ray

Henning Brauer
BS Web Services GmbH

1 Abstract

Modern, digital x-ray machines are pretty complex systems. Different architectures exist - some of them require interconnection of multiple networked system components, but all of them require a connection to the hospital or doctor's office network.

For various reasons, patching a networked medical device to fix security vulnerabilities is not as easy as it might be the case for other IT products. For largely tye same reasons it often takes a long time for legacy network protocols to disappear from the medical device landscape.

Managing security in such environments is a challenge for both the vendors as well as the healthcare providers. As one important layer of defense, a firewall can achieve a reduction of attack surface of legacy devices.

Philips Healthcare, DXR (Diagnostic X-Ray), is offering a firewall for its medical systems for more than 10 years now. It is based on OpenBSD with bridge, pf and an ARP filter extension (ARPF). This system has recently been redone and various improvements, including ARPF, were proposed to the OpenBSD tree and have since then been included.

2 Medical Environment

A digital clinical workflow usually involves managing patient data and scheduling an examination in a radiology information system (RIS). The needed patient- and examination-data is transmitted to the x-ray machine, the operator performs the image acquisitions and on its completion all data is sent to a PACS (picture archiving and communication system) for review and diagnosis as well as long-term archiving.

For old analog x-ray systems (with plain old films), many medical device manufacturers offer digital upgrades. Films are replaced with digital (reusable) cassettes. When such a cassette is exposed to x-ray, it must be inserted into a cassette reader (CR Reader). The reader extracts the image and erases the cassette so it is ready for the next exposure. The read out image is sent to a workstation, where image post-processing happens as well as the linking between the image and the correct patient. CR Readers may be connected anywhere in the network and even though they are legacy devices nowadays, many modern digital systems still support their communication protocols.

Digital medical devices must comply to specific regulation and obtain approvals and certifications before it can be sold and "used on patients". Any design change of a medical device (software or hardware) is guided by processes

and includes extensive system verification and validation, as well as a safety risk management. This is to protect patients and operators from various harms. On the other hand, these well justified efforts hardly permit to release security patches in a timely or frequent manner. On top of that, x-ray systems have a long lifetime (10+ years), usually a lot longer than the support phases of 3rd party software suppliers. So it is just a matter of time until medical systems become unpatchable - unless the vendor and the healthcare provider agree on a business model or service contract which includes major component replacements in order to stay current.

Medical device regulations impose limits on remote management of the devices. For example, the system may not be used on patients while in maintenance mode. The system software may not be updated remotely in a silent way, rather a new software is first uploaded to the system, a person on-site starts the installation (during a non-critical period) and performs a final quality-assurance test.

These circumstances make changes hard and often costly, thus the common incremental approach doesn't really work in this environment.

In medical environments, protection of patient data is of special concern. Other areas see heavy regulation too, i. e. PCI for credit card processors, but protecting electronic patient health information (ePHI) is more critical. Unlike a stolen credit card number, which can be voided, a stolen x-ray image or other ePHI is out there and cannot be voided, because in the majority of cases the affected "medical facts" (like broken bone or breast cancer) cannot be easily changed.

3 architecture of a modern digital x-ray machine

Various types of digital x-ray systems exist to meet the needs of today's clinical workflows. For example a compact, mobile unit is used to perform x-ray at the patient's bed. Fully equipped x-ray rooms allow for a wide range of types of examinations. This includes not only single-shot x-ray but as well fluoroscopy (think x-ray video).

Common architectural elements for those systems are:

- generator and x-ray tube along with circuits and controllers to make sure the patient is exposed to x-ray only with the needed energy and duration.
- An image chain, nowadays usually based on a wireless, digital x-ray detector
- Various hardware in the examination room, like a table, wall stand, foot switch
- A workstation which does image processing, links the images with patients and communicates with RIS and PACS servers in the customers network infrastructure

Some of these systems or components are from third-party suppliers and largely out of the control of the x-ray machine vendor. Due to the relatively small market, high development and certification costs there is little to no competition and thus little to no incentive for third-party suppliers to adapt their components to the vendors wishes.

The system with its components is connected to the hospital's or doctor's offices network - all being in the same layer 2 network.

There is a management system that connects to and configures all other subsystems.

4 Introducing OpenBSD/x-ray

To isolate and protect the system and its internal components from external influences Philips introduced a firewall based on OpenBSD about 10 years ago. Since the system may be shut off by cutting power at any time, a ramdisk based approach has been

chosen. The hardware is an embedded i386 system with, depending on generation, at least 4 Ethernet interfaces.

OpenBSD runs as bridge here, using the bridge(4) subsystem. pf(4) is used to filter network traffic, and the bridge's filter is used for additional layer 2 filtering. The filter rules are set up by the management system.

4.1 arp filtering in the bridge

While pf provides almost everything one could wish for to filter IP traffic, including higher layers like TCP, the bridge filters are kinda limited, and lacked control over arp traffic and thus MAC address learning. Static MAC-IP mappings in all internal systems aren't feasible.

ARP packets are relatively simple and in theory address family independent, but it is really only used for Ethernet and IPv4.

When nodeA with IP address 10.0.0.1 and MAC address 11:22:33:44:55:66 wants to talk to nodeB with IP 10.0.0.2 and MAC 77:88:99:aa:bb:cc, but nodeA doesn't know nodeB's MAC address yet, it sends an arp request (arp who-has) to the broadcast address ff:ff:ff:ff:ff with it's own MAC address in SHA, it's own IP address in SPA, and 10.0.0.2 in TPA. NodeB will reply to the requestor's MAC address, copying SHA and SPA from the request into THA and TPA, it's own MAC address 77:88:99:aa:bb:cc in SHA and it's own IP adress 10.0.0.2 in SPA. NodeA learns the IP 10.0.0.2 to MAC 77:88:99:aa:bb:cc from that.

Many operating systems learn the IP-MAC mapping (fill their arp cache) based on the SPA and SHA of arp requests, even if they are not directly targeted (through a matching TPA) by a given ARP request. That means, ARP spoofing not only works by faking ARP reply packets (when sent to ether broadcast) but also by faking arp requests.

By disallowing arp requests from the outside with any of the inside MAC or IP addresses in SHA and SPA, respectively, and arp replies with any inside IP/MAC in THA/TPA, we can ensure that no inside system is subject to arp spoofing hindering communications between the inside systems. In addition to that, special cases could be filtered when the ARP filter is combined with a filter of the ethernet src or dst MAC address. For example ARP reply to ether broadcast could be a candidate for blocking in some setups.

The bridge filter code has been extended to allow filtering on SHA, SPA, THA and TPA. Assuming em0 as the external interface and an internal IP address 10.0.0.1 and shall be protected, the filter rule would be added to bridge0 using ifconfig:

```
ifconfig bridge0 rule block in \
  on em0 arp spa 10.0.0.1
```

It is also a good idea to prevent any use of the internal MAC addresses - 11:22:33:44:55:66 in this example - from the outside influencing internal communications:

```
ifconfig bridge0 rule block in \
  on em0 src 11:22:33:44:55:66
ifconfig bridge0 rule block in \
  on em0 arp request \
  sha 11:22:33:44:55:66
ifconfig bridge0 rule block in \
  on em0 arp reply \
  tha 11:22:33:44:55:66
```

Reverse arp can be filtered likewise, using the rarp keyword instead of arp.

4.2 Observations from working with the bridge filter code

While it is clear that the bridge needs to die and replaced with switch(4) - that was unfortunately out of scope for this project which was done under contract - a few more observations were made and questions arose.

The bridge filters are very very limited. They can only block and pass, and add a tag that pf can later filter on. There is no logging, and that

offset	length	description	remarks
0	2	Hardware Address Type (HTYPE)	Ethernet: 1
2	2	Protocol Address Type (PTYPE)	IPv4: 0x0800
4	1	Hardware Address Length (HLEN)	Ethernet: 6
5	1	Protocol Address Length (PLEN)	IPv4: 4
6	2	Operation (OPER)	request: 1, reply: 2
8	6	Sender Hardware Address (SHA)	MAC Address
14	4	Sender Protocol Address (SPA)	IP Address
18	6	Target Hardware Address (THA)	MAC Address
24	4	Target Protocol Address (TPA)	IP Address

is really missing to diagnose more complex filters. The configuration through ifconfig is very very basic, the ability to read the bridge filter rules from a file is kind of a hack - the rules are split into words and stuffed in an argv/argc-like array with counter.

The bridge filters are really layer 2 filters - while implemented in the bridge, they don't really belong there, they are useful in scenarios that do not use the bridge at all. More generic layer 2 filters would be desireable.

Layer 2 filters could be implemented completely seperately from the bridge, to be enabled and disabled per interface potentially with a new parser. However, implementing that and implementing proper logging would duplicate a lot of functionality that we already have in pf. Especially for logging we really want to re-use pflog, which, thanks to its bpf logging format, is already suitable for this kind of use.

Can and should we extend pf to be able to filter on ethernet header fields and arp packets? If so, that imposes new questions. pf would have to be able to see packets which are neither IPv4 nor IPv6, these currently do not go to pf at all. We would need a new entry point for ethernet frames - and do we need a seperate layer 2 ruleset? Mixing ethernet header matching with the existing matching logic is tricky - how do we handle non-IP-packets? What do we do with rules that refer to both IP fields and Ethernet header fields when we have a non-IP Ethernet packet, or a non-Ethernet IP packet? Should we make the layer 2 filters part of pf.conf at all, or should we keep that completely seperate?

4.3 Acknowledgements

Holger Mikolon provided help and input with this paper, many thanks.