Global Information Assurance Certification Paper

Technical Implementation of the Critical Control "Inventory of Authorized and Unauthorized Devices" for a Small Office/Home Office

GIAC (GCCC) Gold Certification

Author: Kenton Groombridge, kgroombr@comcast.net
Advisor: Stephen Northcutt
Accepted: 10/26/15

Abstract

There is great value in the proper employment of the Critical Security Controls. The Critical Security Controls are written with terminology that makes them appear applicable only to organizations and other large environments. Implementing the Critical Security Controls can be beneficial to any size network, but can they be applied to a Small Office/Home Office with a limited budget and expertise? This document examines the technical implementation of "Inventory of Authorized and Unauthorized Devices" for a Small Office/Home Office. Topics discussed will be the selection of hardware, evaluation of open-source software and third-party firmware, and custom scripts that function with most modern operating systems.

1. Introduction

The Critical Security Controls are written with wording that makes them appear relevant only for organizations. As per the Critical Security Controls: "The Critical Security Controls are a relatively small number of prioritized, well-vetted, and supported set of security actions that organizations can take to assess and improve their current security state" (Council on CyberSecurity, n.d., p6). One should not be boxed into thinking that these controls are only for organizations, but for all networks no matter how big or small.

When asking users' to define what they use to secure their network, items that come to mind are "Antivirus Software", "Firewall", "Strong Passwords", and other various responses. All answers are relevant to security, but they address security with a hodge-podge approach rather than using an effective, prioritized, and focused set of actions like the Critical Security Controls (Council on CyberSecurity, n.d., p4).

A SOHO should address security like an organization would address security. It starts with a high-level policy. Security policies are published documents that expound the organization's philosophies, strategies, policies and practices with respect to Confidentiality, Integrity and Availability (CIA) of information and information systems (Introduction to Security Policies, Part One: An Overview of Policies | Symantec Connect,. n.d.). A security policy provides a direction to go for the organization. A SOHO should also have a security policy. Rather than start from scratch, a SOHO administrator can use one of the several existing templates and make adjustments necessary to fit the environment. (SANS - Information Security Resources | Information Security Policy Templates, n.d.)

"The Critical Security Controls for Effective Cyber Defense" is a document that provides direction in securing networks. By using knowledge of real world attacks and defenses, it provides guidelines on how to effectively secure networks and implement measures to detect, prevent, or disrupt adversaries in the event they are able to compromise the network (Council on CyberSecurity, n.d., p4). This is a valuable document for a SOHO administrator to utilize since it doesn't dictate what operating systems (OSs), software, or devices to use but provides enough specifics in order to effectively accomplish the task. The SOHO administrator has the flexibility to pick or choose what best fits their requirements within the constraints of their environment and budget.

Kenton A. Groombridge, kgroombr@comcast.net

Implementation of "Inventory of Authorized and Unauthorized Devices" Critical Security Control (CSC), according the Council on CyberSecurity, is in the foundations to success and should considered one of the first controls to implement (Council on CyberSecurity, n.d., p6). For some, this may not be the most important critical control to implement. For example, there is no wireless network connectivity, and all possible ways to connect to the wired network are physically secured. In today's SOHO, this is not the norm as the office and home networks are one in the same.

In a mainstream SOHO, there is wireless network access and there may be several individuals within the SOHO environment that have the permissions and credentials to access (SOHO Wireless Networking. n.d.). These users that access this wireless network (e.g. children and spouses) may give up wireless credentials not necessarily out of maliciousness, but out of helpfulness. Network wall jacks might be installed in every room and when low congestion and ping rates are a priority for gaming, then the kids' friends and friends of friends will be plugging in their cheap dumb switches chaining off to several computers infected from downloading and installing what they believe to be key generators or latest game hacks (Almost all game hacks are infected with malware - Security AffairsSecurity Affairs, n.d.). The proliferation of mobile devices exacerbates the issue as teenagers' jailbreak their phones so they can install applications posted on nefarious third-party sites in an attempt to avoid paying for them, but these applications can come with embedded malware (Android Antivirus: 6 truths about smartphone malware, n.d.).. A good start would be to segregate networks, but Wi-Fi connectivity is always possible within signal range. Even with good physical security there is always the potential of an individual circumventing the controls and connecting a device to the network without the knowledge or approval of the network owner. A good approach to security is to understand that everything is possible and to prepare for the impossible.

In addition to the potential of malicious software piggybacking on systems entering the network, there is always the possibility of a "clean" system entering the network, but has vulnerabilities that may be exploited after the fact. It is just as important to identify and prevent the connection of these systems just the same. All of these are avenues for an adversary to enter the network.

Kenton A. Groombridge, kgroombr@comcast.net

This document will discuss the technical implementation of "Inventory of Authorized and Unauthorized Devices" for a SOHO. It is infeasible to cover every possible configuration for every OS. Support on how to implement a Remote Authentication Dial-In User Service (RADIUS) server, create and implement certificates, configure 802.1x, and so on can be found via numerous freely available documentation. In some instances, tools and processes will be covered at a high level since there is sufficient in-depth documentation. Not all tools will be covered, and those discussed were chosen because they performed the required task. This document does not endorse any tool or application. It is the responsibility of the SOHO administrator to research, select, and implement the appropriate hardware, tools, and applications to satisfy the requirements.

Two major constraints of a SOHO administrator are limited budget and technical expertise (How to sell to the SOHO, n.d.). Implementation may take some expenditure of funds; however, the aim is to stay as close to zero dollars as possible. The ability to implement this control shouldn't take the skills of a rocket scientist. The goal is the make it as easy as possible, but rarely does any security application or device work as required out of the box. The software or device should provide adequate documentation and support that can get it up and running with little difficulties.

2. Sub-Control Categories and Implementation

The Critical Security Controls for Effective Cyber Defense Version 5.1 outlines the "how" at a medium to high level. It starts with the "Quick Win" category which are ways to significantly reduce risk with little financial, procedural, architectural, or technical changes to an environment (Council on CyberSecurity, n.d., p7). It is followed by "Visibility and Attribution" which are ways to improve the processes, architecture, and technical capabilities in order to better detect, identify, and glean information about the event (Council on CyberSecurity, n.d., p7). Next is "Configuration and Hygiene" which reduces the amount and magnitude of security vulnerabilities focusing on protecting against poor security practices by individuals that could give the attacker an advantage (Council on CyberSecurity, n.d., p7). Last is "Advance" sub-controls which use technologies or procedures that may require higher financial, procedural, architectural, or technical changes to an environment (Council on CyberSecurity, n.d., p7). These will be covered in the order listed in the CSC document.

Implementation of individual Critical Security Controls are not isolated processes. For example, if implementing Inventory of Authorized and Unauthorized Software, one application that may be required is a database to maintain the software inventory. This database in turn is an application with accounts, services, and permissions; therefore, Critical Security Controls "Applications Software Security", "Limitation and Control of Network Ports, Protocols, and Service", "Controlled Use of Administration Privileges", and possibly others, must be taken into consideration. In any implementation of a Critical Security Control, it is encouraged to have a list of all critical security controls along with their description and the System Entity Relationship Diagram to visualize how the Critical Security Control interacts with other components. Solutions should take into consideration all other CSCs to ensure a complete, secure, implementation.

3. CSC 1-1 Quick Win

CSC 1-1 is "Deploy an automated asset inventory discovery tool and use it to build a preliminary asset inventory of systems connected to an organization's public and private network(s). Both active tools that scan through network address ranges and passive tools that identify hosts based on analyzing their traffic should be employed" (Council on Cyber Security, n.d., page 9).

For a SOHO, this may appear to be overkill as an inventory of network assets could be manually accomplished by physically accessing the equipment and annotating their information, but even for a SOHO automation is essential. Physical and logical topologies can, and probably will, be different, There may devices hidden away and forgotten that automated scans will find. Also, this inventory will be used as a baseline in which to compare future scans to identify what has changed. It is important to note that systems connected to the network should also include network infrastructure devices such as the cable/DSL modem, wireless router/access-point(s), switches, etc. These should never be overlooked as they, like end devices, can have vulnerabilities that can be exploited.

Information collected from CSC 1-1 feeds into CSC 1-4 which will be discussed in detail later. Before implementation of this step, it may be beneficial to inter-relate these two CSC's and perform them in unison to reduce repetition. Automated inventories may not capture everything, but it is a starting point in which to build upon.

In deciding what information to obtain for the asset inventory, it is best to gather too much information as it is sometimes unknown what information is required until it is needed. As per CSC 1-4, minimum information at this point should include layer 2 Media Access Control (MAC) addresses, layer 3 Internet Protocol (IP) addresses, system name, location, type of device (printer, phone, desktop, laptop, etc.). Some of this will not be obtainable with an automated scan, but it should provide the bulk of it. Since this is a SOHO, there may not be a public network. If there is anything that the public, or external users, must access, it will conceivably be a private system using a static network address translation (NAT) entry configured on the SOHO router mapping an external address/port to an internal address/port.

3.1 Solution for CSC 1-1

There are many ways to perform an automated scan of the network, but many tools that perform this task are outside the budget. There is an open-source tool called Nmap that performs this task very well. Nmap (https://nmap.org/), which is authored by Gordon "Fyodor" Lyon (Gordon "Fyodor" Lyon, n.d.), is often considered the leading tool security professionals use to actively map a network. Nmap is highly configurable with numerous options to fine-tune its operation. It will provide much of the required information for this step. Nmap's configurability and flexibility maybe its own disadvantage which could require some technical expertise; however, there is a wealth of online information.

Nmap is scriptable and can be automated so that little to no human intervention is required once properly configured. Although it provides a considerable amount of information, it does have its limits and may not provide all the required information. Depending on the information needed, it may have to be used with supplementary tools to obtain the remaining information. CSC 1-1's description itself infers that this step is accomplish by utilizing multiple tools. There are many other free or open-source network discovery tools and so don't get boxed into a single solution, although Nmap is hard to beat.

As mentioned, Nmap can perform a various types of scans, but to perform an asset inventory, a ping scan should suffice for the SOHO environment. There are other types of scans that work better for different environments so experimentation should be performed to determine the appropriate scan type. For other types of scans see the Nmap section on host discovery (Host Discovery, n.d.). Additional information is required to satisfy the requirement of other CSCs

such as listening ports. Although this information isn't required for this task, it is prudent to perform scans to capture this information at this time while working with the command line interface. Nmap provides several ways to save its output. It could be redirected to a file, but it has built in output methods to save in various formats, so comparison of formats should be performed to determine the output that meets requirements.

Before starting the process of active scanning, a baseline file must be created of authorized systems with their respective information so that subsequent comparisons can be performed to ensure validity of systems entering the network. In order to be an effective baseline file, it must contain attributes of the scanned devices that are unique to that device. Two major pieces of information are MAC and IP addresses. IP addresses are logical addresses and can be changed, but it is important to know when previously unused IP addresses are detected. MAC addresses tend to be considered hardware addresses; however, they can be easily spoofed. Like IP addresses, previously unknown MAC addresses entering the network must be detected.

When running commands or scripts to create a baseline file, ensure that all devices are connected to the network, powered up, and all wireless devices authenticated to the network to include mobile devices. Doing this ensures that the command or script is able to interact with the end device and records its information in the baseline file.

After the baseline file is created, do not assume that this scan contains a list of trusted, authorized devices. It is possible that there is an unauthorized device which is currently active on the network and listed in the scan. All entries must be scrutinized and validated before retaining them in the authorized baseline list.

For a SOHO, the passive scan may be considered a bit costly and difficult to properly implement. Passive scanning requires the ability to listen on the network and capture information without eliciting a response. In order to effectively perform passive scanning, the capture device must be employed in a location where most, if not all, traffic will traverse. If using IPv4 and all devices are on the same logical layer three segment, then the listening device could be placed anywhere on the network as it would capture all broadcast Address Resolution Protocol (ARP) traffic and eventually discover any device when it attempts to communicate on the network as long as it follows the rules of communication.

Unicast traffic; however, requires knowledge of traffic flow in order to effectively capture. Typically, most devices will eventually attempt to communicate with the way in and out of the network, known as the default-router or default-gateway. A listening device can be placed on the last segment that leads to the default router with the objective to capture this traffic. A problem with most SOHO networks, is that this router usually doubles as the wireless access point, so placing a passive listening device on the cable will not capture the wireless traffic as it bypasses this cable connection.

There are several solutions to meet the intent of this sub-control. One would be to obtain a separate wired router, and configure the wireless router as a wireless access point, so the capture device can be placed in such a way so that all traffic is eventually funneled to the router through the capture device (see Diagram 1: Placement of Passive Capture Device). The goal is the keep cost and complexity low for a SOHO, so another alternative could be to use third party firmware on the wireless router. Recent third-party firmware options such as those from DD-WRT (http://www.dd-wrt.com/) and OpenWRT (https://openwrt.org/) use a recent version of iptables (http://www.netfilter.org) that supports the "--tee" option which provides the capability to perform port-mirroring or Switch Port Analyzer (SPAN). This replicates all traffic passing through the wireless router to an interface so that a capture device can gather all traffic that passes through it (Mike Mackintosh (n.d.). These third-party firmwares are not supported on all devices, so carry out research before making a procurement.

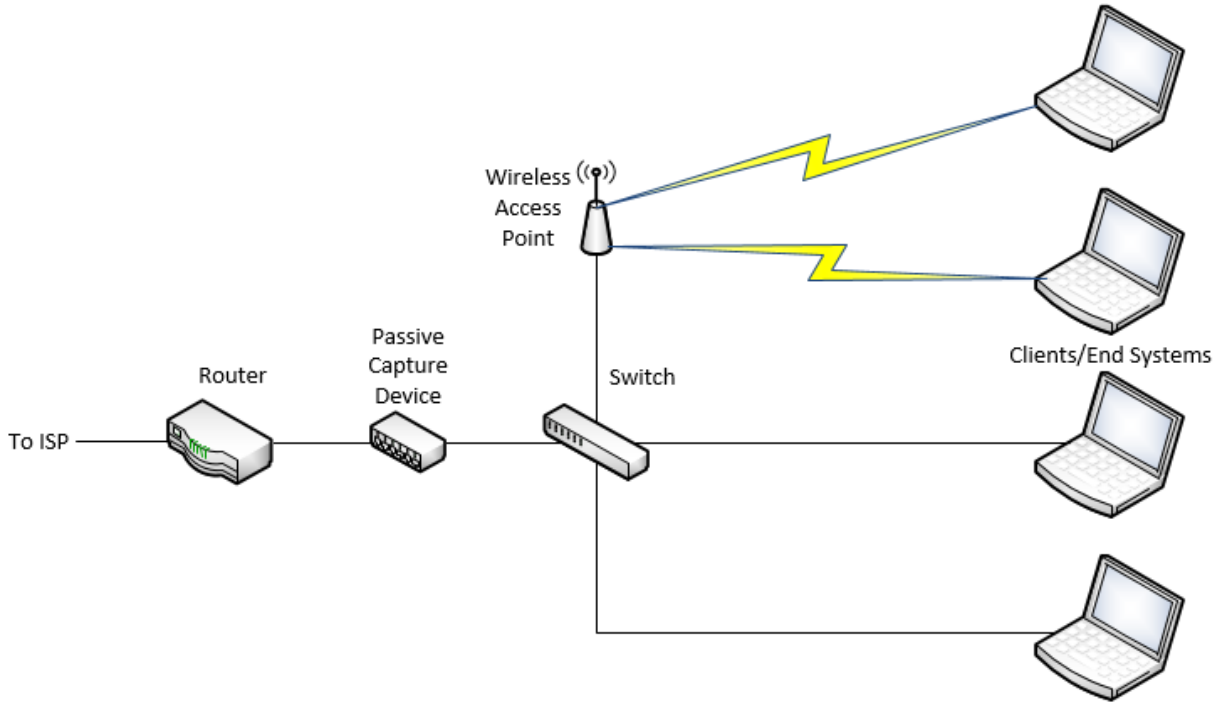

Diagram 1: Placement of Passive Capture Device

It is often best to be paranoid when it comes to security, but sensibly for a SOHO, capturing broadcast ARP traffic should suffice. To save money and complexity, an open-source program called "tcpdump" (http://www.tcpdump.org/), for UNIX, or "windump" (https://www.winpcap.org/windump/), for Microsoft OSs, can be used on the same device that performs active scanning. A separate network interface could be installed and used specifically for passive listening, but isn't required. Tcpdump and windump, using a packet capture driver, will place the interface into promiscuous mode which allows the ability to capture all traffic that the interface senses regardless if it directly addressed to that device, and the device will still capture and process traffic addressed to it.

The ARP cache on the default-router could be valuable as knowing most devices will eventually communicate with it. Although not passive collection, it does provide a method to obtain information about end devices without directly interrogating them. The default-router will store the IP to MAC address binding information in a table after end devices communicates with it, and will maintain this information for a duration called the "ARP cache timeout"

programmed into the device. The ARP cache timeout is normally configurable and may need to be modified to prevent removal of entries before querying the default-router.

As mentioned earlier, it is likely that the default-router for a SOHO is a wireless router. There are third-party firmware for many wireless routers, and most of them provide the ability to remotely access the device with Secure Shell (SSH). Utilizing SSH, a script can be written to regularly access the wireless router and obtain the ARP cache. Through evaluation, it was found that the ARP cache was not flushed for on an evaluation wireless router for nearly ten minutes so as long as the script is scheduled to be executed before that time, it will capture the ARP cache.

There are a couple methods that can be used to perform SSH non-interactively in scripts. One solution is to use sshpass (http://sourceforge.net/projects/sshpass/). Using sshpass requires the SSH password to be entered in the script so that the script will run without intervention, but this is highly discouraged since the password in the script will be clear text and thus susceptible to theft. The second solution is to enter the authorized SSH key on the wireless router to trust the system executing the script. DD-WRT, Tomato, and other third-party firmwares support SSH keys.

4. CSC 1-2 Quick Win

CSC 1-2 is "Deploy dynamic host configuration protocol (DHCP) server logging, and utilize a system to improve the asset inventory and help detect unknown systems through this DHCP information" (Council on Cyber Security, n.d., page 9).

Implementing DHCP presents a challenge to control connectivity to the network since a system or device can easily obtain an IP address. Connecting a device with DHCP enabled is almost like saying "Hey, you are a device that supports DHCP, so here is an IP address and welcome to the network." Employing DHCP can make it difficult to track down systems and devices as they will obtain an address via DHCP as long as they connect to a segment of the network that supports it.

4.1 Solution for CSC 1-2

DHCP is optional. For a small network like a SOHO network, DHCP can be disabled and IP addresses can be assigned statically, but this reduces the convenience that DHCP provides, and

even if not using DHCP, this doesn't preclude an individual from determining the address range in use and entering an available IP address statically on their system, so disabling DHCP may not be the best solution. When using DHCP, it is good to configure reservations on the DHCP server so that systems and devices obtaining a DHCP address will always be offered the address that is reserved for that device. This reduces the effort involved in monitoring the network as it provides consistency.

For many SOHO environments, the device that provides DHCP services is the wireless router, but with the stock firmware there is usually no configurable option for remote logging which makes it difficult to satisfy this sub-control. As mentioned earlier, third party firmware such as DD-WRT has the capability to utilize a remote syslog server. UNIX derivatives such as Linux support syslog, and there are free options for Windows platforms such as "Syslog for Windows" (http://syslog-win32.sourceforge.net/), "Kiwi Syslog Server" (http://www.solarwinds.com/products/freetools/free-kiwi-syslog-server.aspx) by Solarwinds, and many others.

Once a syslog server is installed, configured and the devices that support remote logging are configured to transmit their logs to the syslog server, then a method must be put in place to examine the logs checking for unauthorized devices receiving DHCP leases. Filtering commands can extract the DHCP messages by looking for the keyword "DHCP". These in turn, can be fed into scripts where the MAC and IP addresses can be compared to the baseline.

Another method to obtain the DHCP binding information is to use SSH in a similar fashion where the ARP cache can be extracted from the wireless router. The wireless router, since it is acting as a DHCP server in a SOHO environment, maintains the DHCP binding information, and if using SSH capable third-party firmware on the wireless router, then this information can be easily retrieved.

A script can be written to periodically access the wireless router using SSH and retrieve the table of which devices have obtained an IP address from the wireless router via DHCP. This in turn can be compared to the baseline file to verify the device. Gleaning MAC addresses from the DHCP bindings table will not detect MAC address spoofing which, as mentioned earlier, is trivial to employ. Also, static addresses assigned to end devices will prevent them from requesting a DHCP address; therefore, they will not be listed in the DHCP binding table.

5. CSC 1-3 Quick Win

CSC 1-3 is "Ensure that all equipment acquisitions automatically update the inventory system as new, approved devices are connected to the network" (Council on Cyber Security, n.d., page 10).

5.1 Solution for CSC 1-3

The SOHO administrator is also the owner and operator so this is solved with procedural solutions to ensure once a piece of equipment is procured, then its information must be added to the baseline file to ensure that future scanning and monitoring of the network does not flag the newly acquired device or system as unauthorized. There should be a mechanism to add the information from newly acquired systems and devices to the baseline file.

Although not directly written in this CSC sub-control, it is equally important to remove entries for retired devices or systems from the baseline file. The additional information in the file may appear to be benign at first thought. Mentioned previously in CSC sub-control 1-2, it was noted that monitoring DHCP information doesn't catch a device where the MAC address is spoofed to appear like an authorized device or system on the network. If MAC address spoofing of the retired system or device continues to occur, removing its entry from the baseline file will now generate an alert. Without proper management of the baseline file, unauthorized systems and devices performing MAC address spoofing would not be noticed.

6. CSC 1-4 Visibility and Attribution

CSC 1-4 is "Maintain an asset inventory of all systems connected to the network and the network devices themselves, recording at least the network addresses, machine name(s), purpose of each system, an asset owner responsible for each device, and the department associated with each device. The inventory should include every system that has an Internet protocol (IP) address on the network, including but not limited to desktops, laptops, servers, network equipment (routers, switches, firewalls, etc.), printers, storage area networks, Voice Over-IP (VoIP) telephones, multi-homed addresses, virtual addresses, etc. The asset inventory created must also include data on whether the device is a portable and/or personal device. Devices such as mobile phones, tablets, laptops, and other portable electronic devices that store or process data must be

identified, regardless of whether they are attached to the organization's network" (Council on Cyber Security, n.d., page 10.)

6.1 Solution for CSC 1-4

The words "at least" in this sub-control hints that there should be additional information recorded, but each environment will dictate what is important to annotate. Some of this is a repeat of information recorded sub-control 1-1. From experience, some additional information that should be documented are MAC addresses, make/model/serial numbers, RAM capacity, disk/storage space, and other items specific to the hardware. Although some of this information isn't related to security, it is important to have in the event of theft, or if determining if the systems are compatible in with the latest OS or application. This list should be printed and stored in a safe location in the event that the digital copy is lost.

7. CSC 1-5 Configuration and Hygiene

CSC 1-5 is "Deploy network level authentication via 802.1x to limit and control which devices can be connected to the network. The 802.1x must be tied into the inventory data to determine authorized versus unauthorized systems" (Council on Cyber Security, n.d., page 10).

The benefit of using 802.1x is that a device can attempt to network, but authentication takes place at layer 2 (frame). Until the end system or device authenticates, there will be no traffic that will pass through the authenticator. The authenticator in this case is the switch or wireless router/access-point.

7.1 Solution for CSC 1-5

Implementing network level authentication with 802.1x may take some effort, but it is reasonably possible for a SOHO. Most modern wireless routers include this capability, although in the configuration it may not state "802.1x". If the wireless router has WPA/WPA2 Enterprise, then it has 802.1x capability; however, it probably supports 802.1x only for wireless connections. Implementing 802.1x with via a wired device will require the procurement of a switch that supports 802.1x. 8-port wired switches that support 802.1x can be found for around $100 and 24-port switches for close to $150. Another option is buying used equipment. A Cisco Catalyst 3560 48 port Power over Ethernet (PoE) switch can be procured for less than $50

including shipping, and these have features and capabilities well beyond what is required for a SOHO.

Employing 802.1x network level authentication requires more than just 802.1x compatible hardware. The end node has to support 802.1x through what is called a supplicant which is usually software. An authentication server is also required which is normally an application. Discussing the details of 802.1x are is beyond the scope of this document and there are numerous resources that cover it in detail; however, some detail of implementation for a SOHO will be examined.

The authentication server is the system that ultimately authenticates the client that attempts to connect to the network either wirelessly or physically. A RADIUS server is normally used as an authentication server. The RADIUS server can maintain a database of user accounts, but also can reach back to other centralized authentication databases such as Active Directory (AD). Since this is a SOHO, there probably isn't an AD infrastructure, so the user account database will likely be local and maintained by the RADIUS server.

There are many RADIUS server options, but very few of them will be in the price range of a SOHO. One option, if using a UNIX derivative such as Linux, is FreeRADIUS (http://freeradius.org/). As the name implies, it is free and includes RADIUS server, a BSD licensed client library, a PAM library, and an Apache module (FreeRADIUS, n.d.). The FreeRADIUS documentation is well written and the server is relatively easy to setup and configure using the default configuration files. The only changes required to get 802.1x operational, in addition to the configuration changes to the client, are the addition of username/password combinations in the "users" file, and an entry in the "clients.conf" file. This entry in the clients.conf file configures the authenticator, which is the switch or wireless router, on what shared secret is used for the communication.

For the 802.1x supplicant, Windows natively supports 802.1x, and flavors of UNIX can use software called wpa_supplicant (http://w1.fi/wpa_supplicant/). For UNIX, the name "wpa_supplicant" sounds like it exclusively with wireless, but it also supports wired 802.1x connections. There is an older program called XSupplicant which was designed to be a wired 802.1x supplicant (http://open1x.sourceforge.net/), but it hasn't been supported since 2010 and is considered deprecated.

Should one choose to use a RADIUS server on Windows, there are a couple "free" options. One option is an older version of FreeRADIUS has been ported to Windows (http://freeradius.net/), but unfortunately it is based on the older unsupported 1.X FreeRADIUS codebase. It operates correctly on Windows 7 32-bit and 64-bit versions, so it is expected to work on newer versions of Windows as well. The configuration of the "users" and "clients.conf" file of FreeRADIUS 1.X is identical to that of the FreeRADIUS 2.2.5 for UNIX flavors. One problem encountered with FreeRADIUS for Windows, was when running it in service mode, it fails to authenticate, but when running in debug mode, it authenticated correctly. This was a challenge as there was no resolution to this found via Internet searches. With some examination and troubleshooting, the solution found to this is to create a new user account with a new name such as "FreeRADIUS", and then configure the service to run as this new user. This new user account should only be used by the FreeRADIUS server, and not as a normal user. With respect to the other CSCs, some locking down of this user account should be performed such as disabling interactive-logon, a complex password, and other items to secure this account.

A RADIUS server called "TekRADIUS" (http://www.kaplansoft.com/tekradius/) is another option for Windows. It comes in two flavors, a free version and a commercial version with the commercial version containing additional features (KaplanSoft - Art of Software (TekRADIUS), n.d.). If only requiring the ability to use 802.1x with username/password authentication, then the free version TekRADIUS will suffice. Windows users will find the Graphical User Interface (GUI) a bit more comfortable to use than editing the text files required for FreeRADIUS. The TekRADIUS documentation is very well written and the free version has excellent support. An issue was encountered with TekRADIUS and after posting the issue in the TekRADIUS forums (http://forums.tekradius.com/), it was address and resolved by the TekRADIUS author in less than 24 hours.

8. CSC 1-6 Configuration and Hygiene

CSC 1-6 is "Configuration and Hygiene, Deploy network access control (NAC) to monitor authorized systems so if attacks occur, the impact can be remediated by moving the untrusted system to a virtual local area network that has minimal access" (Council on Cyber Security, n.d., page 10).

Sometimes the concept of NAC is confused with 802.1x. Although many NAC applications use 802.1x as a component of the total NAC solution, it is only a part of the whole system. NAC isn't a standard and so when choosing a NAC solution; it may involve buying into a single vendor as the NAC components from different vendors many not interoperate.

8.1 Solution for CSC 1-6

There are several free open-source versions of NAC available: PacketFence (http://www.packetfence.org/), FreeNAC (http://freenac.net/), OpenNAC (http://www.opennac.org/), and possibly others. All of these can be used with a SOHO, but one stands out for ease of use and that is a version of PacketFence called Zero Effort NAC (ZEN). PacketFence ZEN is packages as an Open Virtual Appliance (OVA), which allows many visualization software applications to open and execute it. It is a complete system running on a Linux based OS and by using the included wizard, it can be configured quickly to get NAC up and running on the network.

As with 802.1x, any of these NAC solutions will require smart switches and other upgraded hardware. Using an evaluation SOHO network configured with a second hand Cisco 3560 switch and a wireless router modified with Tomato firmware (http://tomato.groov.pl/), NAC, using PacketFence ZEN was operational within an hour. The other free NAC solutions were not evaluated, and in all fairness they should be tested before taking a decision.

Although not a NAC solution, there are some things that provide NAC like functionality if using switches that support Simple Network Management Protocol (SNMP) link change or MAC address change Management Information Base (MIB) traps. SNMP traps can be extremely valuable for network monitoring. Their value is that they are sent in near real-time to when the event occurs. With respect to other CSCs, if implementing SNMP, use SNMP version 3 as it supports confidentiality and integrity. . A free SNMP server is available for UNIX and Windows called Net-SNMP (http://www.net-snmp.org/), which will allow the receipt and logging of SNMP messages.

New switches were found that support SNMP traps for less than $150, so it is within the price range for a SOHO, but again, research to ensure it supports the required MIBs before

purchase. The Cisco 3560 switch used for this evaluation supports both of these MIB notifications.

If given the support for both MIBs, the one to implement is MAC address change notification. If the switch learns or flushes a MAC address from its Content Addressable Memory (CAM) table, then an immediate notification is sent to the SNMP server with information such as MAC address, interface association, and switch IP address. With this information, a device entering or leaving the network can be quickly hunted down.

If link change notification messages are the only option, it will work well for a SOHO as it should be uncommon for devices to be plugged in or removed from the network. Although the link change notification messages will not give the MAC address of the device to determine if it is an authorized or unauthorized device, it will give the switch IP address and interface where the connection is made so that it can be tracked down and validated manually.

9. CSC 1-7 Advanced

Go through an overview of this.

CSC 1-7 is "Utilize client certificates to validate and authenticate systems prior to connecting to the private network" (Council on Cyber Security, n.d., page 10).

This may sound like a daunting task, and the Council on CyberSecurity doesn't help by calling it "advanced", but it reality this takes little additional time and effort to implement for a SOHO assuming 802.1x is currently configured with username/password authentication.

9.1 Solution for CSC 1-7

Although the word "certificate" makes it sound like a Certificate Authority (CA) has to be stood up, and operational, for a SOHO environment, locally generated, and trusted, certificates can be created and used for this purpose. The process is well-documented using Internet resources. The FreeRADIUS site has documentation on getting certificate authentication functional which worked correctly with FreeRADIUS version 2.2.5 (Deploying RADIUS: Configuring EAP, n.d.).

OpenSSL (https://www.openssl.org/) is the open source tool that can be used to generate and sign certificates. For this deployment, OpenSSL is used to generate a self-signed certificate

which will act like a root CA. This self-signed certificate is used to sign the server certificate used on the RADIUS server, and the client certificates used on the end devices.

For Windows users TekRADIUS also has the capability to use certificate authentication as well. There is a Windows port of OpenSSL, which was used to create certificates for the evaluation network, but the author of TekRADIUS has a product called "TekCERT" (http://www.kaplansoft.com/tekcert/) that is a Certificate/Certificate Signing Request (CSR) Generator and Signing Tool which runs under all modern versions of Windows. Like the TekRADIUS server, there is a free and paid version with the free version having some limitations, but it was sufficient to evaluate the usefulness of this tool. Windows users will find the GUI a welcome change over using the command-line based OpenSSL.

All end devices and systems must also be modified to support certificate based authentication and have certificates installed. Modern Windows systems support this natively, and flavors of UNIX can use wpa_supplicant to provide this support.

10. Utilizing Scripts

Scripting can be a method to provide capabilities that may otherwise be expensive or non-existent for a SOHO. Many of these processes could be performed manually, but one of the goals of the CSCs is to automate defenses (Council on CyberSecurity, n.d., p5). The hardware and software implemented in a SOHO was likely acquired because it provided services or functions required to carry out everyday business functions or activities, not because it added security value. Scripting can provide the automated custom security functionality that, with some time and expertise, can work with a considerable amount of discontinuous existing hardware and software employed in a SOHO.

The appendix contains two scripts. As discussed throughout this document, scripts and automation can perform many of the requirements of this CSC. Portions of the scripts found in the appendix have been touched upon briefly in many sections of the document. The scripts do not perform every function to meet every requirement of this CSC, but they do perform many of the sub-controls. They are designed to show how ingenuity and scripting can tie together multiple techniques in gathering, evaluating, and providing important information relevant to CSC1.

Nmap version 6.49BETA5 was used to perform the active scanning portion from CSC 1-1. The output format chosen is Extensible Markup Language (XML). XML was selected since the scripting language used is Perl 5.22.0 (https://www.perl.org/) as it is supported on nearly every OS and there is a module for Perl that simplifies parsing an XML Nmap scan (http://search.cpan.org/~apersaud/Nmap-Parser/Parser.pm). Using XML output for Nmap and Perl for scripting, produces portable scripts that should work with nearly any system with little to no modifications.

From this point forward, Nmap, Perl, and the Nmap-Parser Perl module must be installed in order to use these scripts. Find or create a directory on the system and make certain that all scripts and files are retained and executed from this directory. This is to ensure that all output is saved in this directory as the scripts depend upon this. At this time, all IP addresses are assumed to be IPv4 addresses, but with some massaging these scripts and commands can be adjusted to work with both IPv4 and IPv6 addresses. These scripts require elevated privileges since they perform functions that require special access such as scanning and enabling promiscuous mode on interfaces, so run them as the root user, or create a special user with appropriate privileges. All scripts can be found in the appendix.

10.1 Generating the Baseline

The getbaseline.pl script, found in the appendix, creates a baseline by scanning systems utilizing Nmap, which in turn saves the output in XML format. The script then parses the XML and converts the information into a Comma Separated Values (CSV) file named "baseline.csv". A CSV file as it is sufficient for storing the information for a SOHO and can be opened and edited with a spreadsheet program such as Microsoft Excel, LibreOffice Calc or any text editor. This script needs elevated privileges or executed with the proper capabilities (Running nmap as an unprivileged user, n.d.). This script will produce a lot of information but only the information that pertains to an asset inventory will be collected. The creation of the baseline scan file will be used to compare future connection attempts to ensure that only valid devices and systems are allowed and to alert on those that are not listed in the baseline.csv file.

The getbaseline.pl script checks to see if a value was returned for each column, and if there is no value, it enters in the keyword "UNDEFINED". As seen in the example baseline.csv file below, the first line with the IP address of 192.168.0.10 has the MAC address, IP address, and

MAC vendor column listed as UNDEFINED. This is because the script is executed from that system and Nmap doesn't have to ARP for its own address since it is the source device. Because the MAC address is unknown, then subsequently the MAC vendor is UNKNOWN. The last two lines list the resolved hostname as UNDEFINED. This is not a concern, as there will be instances where a hostname isn't configured or resolvable, especially in this case since the last two entries are networking devices and not hosts.

```
192.168.0.10,UNDEFINED,UNDEFINED,sys1.hsd1.ca.comcast.net
192.168.0.1,48:F8:B3:CB:B6:95,Cisco-Linksys,Linksys04515
192.168.0.13,C0:3F:0E:C5:E8:CD,Netgear,sys2.hsd1.ca.comcast.net
192.168.0.15,00:26:98:98:CE:C0,Cisco Systems,UNDEFINED
192.168.0.2,48:F8:B3:C9:AE:BB,Cisco-Linksys,UNDEFINED
```

Figure 1: Example baseline.csv File

10.2 Scanning Script

The "scan.pl" script, found in the appendix, is designed to perform a scan at regular intervals and compare its results to the baseline. It has several scans and enumeration techniques which can be enabled/disabled as needed. Each technique is designed to determine end devices attached to the network and compare them against the previously acquired baseline.csv file. If a device is discovered that isn't annotated in the baseline.csv file, then it will provide an immediate alert. The interval that this script is executed should be less than the optimum discovery time of an unauthorized connection and often enough to discover a connection even if is connected for short period of time. This can be adjusted in the script by changing the "INTERVAL" constant near the top of the script.

Many parts of the scan.pl script rely on SSH to remotely execute or extract information from devices. It is not written to utilize the "sshpass" command, but could be modified to do so. In order for it to function without interaction, trusted SSH keys must be installed on the remote enumerated devices.

10.3 Performing the Active Scan

Setting the NMAP constant near the top of the script to a numeric "1" will allow this script to satisfy the requirements of an active scan. Like the technique used to obtain the baseline.csv file,

Nmap is used to interact with the end devices and obtain their information. If using the Nmap active scan, edit the NETWORK constant and set it to the network/subnetwork address of the target network. The IP and MAC addresses are obtained and are used to validate the end device. These addresses are compared against those recorded in the baseline.csv file to determine discrepancies. Anything that doesn't match completely will be saved in the file "suspicisous.csv" so that it is recorded and submitted to the SOHO administrator. To disable the active scan functionality in the scan.pl script, set the NMAP constant to a numeric "0".

10.4 Performing the Passive Scan

The Perl script "getarp.pl" in the appendix optionally uses tcpdump to capture ARP traffic, specifically ARP reply traffic, which contains the IP address with corresponding MAC address. In order to obtain only the ARP replies, the Berkeley Packet Filter (BPF) added to the tcpdump command looks for ARP messages with the opcode equal to "2" which is an ARP reply. The opcode is a two byte value at offset of 20 in the Ethernet frame. The tcpdump switches used are "-t" to disable timestamps as they aren't required for this purpose, and "-n" to disable IP address to name resolution as the numeric IP address is required to compare to the same in the baseline.csv file. In order to obtain immediate updates to this file, the output buffer is set to "0", or "do not buffer", with the stdbuf Linux command. Without this command, the output will be flushed to the file only once the buffer is full. Essentially, this could cause entries to be written to the file long after the ARP reply was received. The intention of the critical controls is to have timely notification, so this file must be updated immediately.

```
#stdbuf -o0 tcpdump -tn arp and ether[20:2] = 2 > tcpdump.txt
```

Figure 2: tcpdump command to capture ARP replies

This command is forked to the background, as it has to run continuously. In order to prevent the tcpdump.txt from continuously growing, it is truncated by overwriting it with"/dev/null" once the information is gleaned from it. Notice that the output is redirected to a file instead of using the "-w" switch that writes packets to a file. The binary packet contents are not required for this, just the messages showing the IP and MAC addresses necessary to compare to the baseline.csv file to determine validity of the system.

```
ARP, Reply 192.168.0.14 is-at 1c:3e:84:d4:8e:d7, length 46
ARP, Reply 192.168.0.15 is-at 00:26:98:98:ce:c0, length 46
ARP, Reply 192.168.0.12 is-at d0:4d:2c:39:ff:b1, length 46
ARP, Reply 192.168.0.13 is-at 84:38:38:23:ad:8c, length 46
ARP, Reply 192.168.0.11 is-at 00:23:15:75:ba:74, length 46
```

Figure 3: Example of ARP reply tcpdump command

10.5 Gleaning the ARP Cache from the Default Router

The script, "scan.pl" in the appendix has the capability to use SSH to remote into a device, extract the ARP cache, and compare it to the entries listed in the baseline.csv file. There is a constant called "ARP" near the top of the script and setting this constant to a numeric "1" enables this feature, and setting it a numeric "0" disables this feature. This script was written for Tomato firmware, but it should work for other third-party firmwares as currently written or with path/filename modifications. If enabling this feature, the DEFAULT_ROUTER constant must be changed to the IP address of the device to glean the ARP cache, which will probably be the default-router. As long as the scan.pl script is executed within an interval less than the time the ARP cached is flushed, then it will capture information on all devices that have communicated via ARP with the default-router.

Using a router modified with Tomato firmware, using SSH to remotely access the wireless router and executing the "arp" command provides its ARP cache which is saved to a file. The ARP table retrieved by this method contains the IP address to MAC address bindings necessary to compare them against the baseline file to determine validity of the system. The hostnames in the below output were modified for readability.

```
android-X (192.168.2.22) at F8:A9:D0:0D:82:3E [ether] on br0
sys1-X (192.168.0.10) at D0:50:99:40:EA:00 [ether] on vlan2
sys2-X (192.168.0.1) at 48:F8:B3:CB:B6:95 [ether] on vlan2
```

Figure 4: Example ARP Table Retrieved Via SSH

10.6 Gleaning the DHCP Bindings Table

The script, "scan.pl" in the appendix has the capability to use SSH to remote into devices to glean the DHCP binding information directly from the device and compare it to the baseline.csv file. This script was written for Tomato firmware, but it should work for other third-party firmwares as is or with a path/filename modification. This is an optional feature that is enabled or disabled by a constant called "DHCP" near the top of the script. Setting this constant a numeric "1" enables this feature, and setting it a numeric "0" disables this feature. If enabling this feature, the DHCP_SERVER constant must be changed to the device responsible for DHCP services.

The DHCP binding table pulled by using SSH contains both the IP address to MAC address bindings required to check them against the baseline file. The hostnames in the below list were modified and remaining hexadecimal output was truncated for readability. Through some recognition, the first column "86400" is the lease time in seconds which equates to one day. The MAC addresses in the output of the DHCP bindings table uses lowercase hex digits while the output from previous commands used uppercase. MAC address are expressed in different formats depending on the system. Hexadecimal digits can be upper or lower case, and the character used to separate each octet can be a colon ":" or a dash "-". In order to compare MAC addresses, normalization of the address format must be performed, or the command comparing the addresses must be aware of the encoding.

```
86400 f8:a9:d0:0d:82:3e 192.168.2.22 android-X ff:d0:.....
86400 d0:50:99:40:ea:00 192.168.2.25 sys4-X ff:d0:.....
86400 48:f8:b3:cd:b6:95 192.168.2.26 sys5-X ff:d0:.....
```

Figure 5: Example DHCP Mapping Table Retrieved Via SSH

10.7 Gleaning the SNMP Trap Messages

SNMP messages are usually sent to syslog. For this solution, the net-snmp (www.net-snmp.org/) package was installed, and the snmptrapd.conf file was modified so that SNMP traps are logged to the same directory as the scripts in its own file called "snmptrapd.log". SNMP message entries can be somewhat cryptic, and can be multi-line, but with some analysis they can

be dissected and their format and meaning interpreted. The below entries are produced by a Cisco 3560 switch configured with MAC address change notifications. The entries display the MAC address as part of its Hex-STRING output "`28 D2 44 36 D8 14`", so this MAC address format will need to be manipulated so it is in the same format listed in the baseline file in order for it to be compared to determine if this is an authorized or unauthorized system. The first byte in the hex string is indicative of the device learning or flushing the MAC address. If this byte is a "`01`", then it is a learn event, if it is a "`02`", then it is a flush event.

```
2015-10-03 14:07:42 192.168.0.11(via UDP: [192.168.0.11]:55777-
>[192.168.0.10]:162) TRAP, SNMP v1, community public
     SNMPv2-SMI::enterprises.9.9.215.2 Enterprise Specific Trap
(1) Uptime: 13 days, 23:26:25.72
     SNMPv2-SMI::enterprises.9.9.215.1.1.8.1.2.1 = Hex-STRING:
01 00 01 28 D2 44 36 D8 14 00 0C 00     SNMPv2-
SMI::enterprises.9.9.215.1.1.8.1.3.1 = INTEGER: 120758572
```

Figure 6: Syslog entry for SNMP MAC Address Learned Event

```
2015-10-03 14:07:44 192.168.0.11(via UDP: [192.168.0.11]:55777-
>[192.168.0.10]:162) TRAP, SNMP v1, community public
     SNMPv2-SMI::enterprises.9.9.215.2 Enterprise Specific Trap
(1) Uptime: 13 days, 23:26:27.73
     SNMPv2-SMI::enterprises.9.9.215.1.1.8.1.2.1 = Hex-STRING:
02 00 01 28 D2 44 36 D8 14 00 0C 00     SNMPv2-
SMI::enterprises.9.9.215.1.1.8.1.3.1 = INTEGER: 120758773
```

Figure 7: Syslog entry for SNMP MAC Address Removed Event

These messages are sent from a layer 2 switch so it doesn't comprehend the end node layer 3 (IP) address. There are IP addresses in the messages, but these are the IP addresses of the switches that generated the message. This IP address is important to have in log output because if there are multiple switches, then it will uniquely identify the switch where the event took place.

10.8 The Script in Practice

When the script is executed, it will execute those actions enabled and configured at the top of the script. It will perform each section sequentially, record its findings in the file suspicious.csv and send that via a chosen alert method. Currently it is set to display the messages to the console, but can be modified to alert via many different methods. If there is nothing written to the suspicious.csv file, then there is no alert. The script loop delay is set by changing the INTERVAL constant, in seconds. This value should be set to equal or less than the optimum detection time. It will loop continuously until it is forcefully closed. Below lists a suspicious.csv file displaying the differences found with detection method and other pertinent information:

```
192.168.0.16,68:17:29:D4:E3:E2 - NMAP Scan - New IP and MAC Address
192.168.0.22,F8:A9:D0:0D:82:3E - ARP Cache - New IP and MAC Address
192.168.0.10,D0:50:99:40:EA:00 - ARP Cache - New MAC Address
192.168.0.22,f8:a9:d0:0d:82:3e - DHCP Table - New IP and MAC Address
192.168.0.10,d0:50:99:40:ea:00 - PASSIVE - New MAC Address
28:D2:44:36:D8:14 - TRAP - New MAC Learned at Switch IP 192.168.0.11
28:D2:44:36:D8:14 - TRAP - New MAC Flushed at Switch IP 192.168.0.11
```

Figure 8: Example suspicious.csv file

The goal of this script is to take all acquired information, compare it against the baseline, and provide meaningful output. The suspicious.csv file contains the IP and MAC addresses of systems that do not match entries present in the baseline file along with the detection method, and the reason it was flagged. Editing the script is relatively straight forward and can be done to change output order or wording.

Notice it is possible for IP and MAC addresses to be listed twice or more. This is not an issue as they were found by different methods that corroborate the evidence that there definitely is a system or device which IP or MAC address doesn't match a known system in the baseline.csv file. Also note the "TRAP" messages show that a device was inserted and removed, but none of the other methods caught this. SNMP Traps are very fast, and in this case, the device was plugged in for a fraction of a second, then immediately removed. The device was able to get at least one frame through the switch for it to learn its MAC address, but it never obtained a DHCP lease; therefore, it never issued an IP address required for other methods to detect it.

A goal of the CSCs is to receive and act upon this information as quickly as possible prescribed by standards. The SOHO administrator must examine each entry to determine is validity, and if valid, its information should be placed in the baseline.csv file to prevent future alerts. By not adding authorized systems to the baseline.csv document, then the additional authorized entries listed in the alerts will increase the likelihood of human error inadvertently not noticing an unauthorized entry.

11. Conclusion

The Critical Security Controls are a layered security approach to cover all the gaps in a complete security solution. In the event that an adversary or malicious code defeats one control, then another control could detect and could possibly block it. In a similar manner, each individual control has various overlapping mechanisms to compliment and or supplement other controls so that they work together and ensure if any method is defeated, another method will succeed. It has been verified that the implementation of the Critical Security Control "Inventory of Authorized and Unauthorized Devices" for a home or small office is possible, but involves time, expertise, and money.

12. References

Almost all game hacks are infected with malware - Security AffairsSecurity Affairs. (n.d.). Retrieved from http://securityaffairs.co/wordpress/13640/malware/almost-all-game-hacks-are-infected-with-malware.html

Android Antivirus: 6 truths about smartphone malware. (n.d.). Retrieved from http://phandroid.com/2014/05/06/android-virus-malware-scan/

Council on CyberSecurity (n.d.).The Critical Security Controls for Effective Cyber Defense Version 5.1 Retrieved from http://www.counciloncybersecurity.org/critical-controls/

Deploying RADIUS: Configuring EAP. (n.d.). Retrieved from http://deployingradius.com/documents/configuration/eap.html

FreeRADIUS: The world's most popular RADIUS Server. (n.d.). Retrieved from http://freeradius.org/

Gordon "Fyodor" Lyon. (n.d.). Retrieved from http://insecure.org/fyodor/

Host Discovery. (n.d.). Retrieved from https://nmap.org/book/man-host-discovery.html

How to sell to the SOHO. (n.d.). Retrieved from http://www.infordiss.com/Pages/How-to-sell-to-the-SOHO.aspx

Introduction to Security Policies, Part One: An Overview of Policies | Symantec Connect. (n.d.). Retrieved from http://www.symantec.com/connect/articles/introduction-security-policies-part-one-overview-policies

KaplanSoft - Art of Software (TekRADIUS). (n.d.). Retrieved from http://www.kaplansoft.com/tekradius/

Mike Mackintosh (n.d.) Packet Cloning With iptables Retrieved from https://www.mikemackintosh.com/packet-cloning-with-iptables/

netfilter/iptables project homepage - netfilter/iptables - Patch-o-Matic Listing - external. (n.d.). Retrieved from http://www.netfilter.org/projects/patch-o-matic/pom-external.html

Running nmap as an unprivileged user. (n.d.). Retrieved from https://secwiki.org/w/Running_nmap_as_an_unprivileged_user

SANS - Information Security Resources | Information Security Policy Templates |. (n.d.). Retrieved from http://www.sans.org/security-resources/policies/)

SOHO Wireless Networking. (n.d.). Retrieved from http://www.l-com.com/content/Article.aspx?Type=N&ID=167

12. Acronyms

ARP	Address Resolution Protocol
BPF	Berkeley Packet Filter
CA	Certificate Authority
CAM	Content Addressable Memory
CIA	Confidentiality, Integrity, and Availability
CSC	Critical Security Control
CSV	Comma Separated Values
DHCP	Dynamic Host Configuration Protocol
DSL	Digital Subscriber Line
GUI	Graphical User Interface
IP	Internet Protocol
MAC	Media Access Control
MIB	Management Information Base
NAC	Network Access Control
NAT	Network Address Translation
OS	Operating System
OVA	Open Virtual Appliance
PoE	Power over Ethernet
RADIUS	Remote Authentication Dial-In User Service
RAM	Random Access Memory
SNMP	Simple Network Management Protocol
SOHO	Small Office/Home Office
SPAN	Switch Port Analyzer
SSH	Secure Shell
VoIP	Voice over Internet Protocol
XML	Extensible Markup Language
ZEN	Zero Effort NAC

13. Appendix

Perl script getbaseline.pl

```
#!/usr/bin/perl
# baseline.pl
# Written by Kenton Groombridge
# THIS SOFTWARE COMES WITH ABSOLUTELY NO WARRANTY! USE AT
# YOUR OWN RISK!
use strict;
use warnings;
use Nmap::Parser;

# This is the filename of the baseline file.
use constant FILENAME => "baseline.csv";
# Make this value the network address of the network to scan.
use constant NETWORK => "192.168.0.0/24";

# Set up variables and other items required for the script
my $my_hostname;
my $my_ipv4_addr;
my $my_mac_addr;
my $my_mac_vendor;
my $np = new Nmap::Parser;

# Setup the Nmap command line with appropriate network address defined
# above.  Send default output to /dev/null as it isn't needed
my $nmap_command_line = ("nmap -sn -oX nmap.xml ".NETWORK." >
/dev/null");

# Exit if the baseline document exists as to not inadvertently
# overwrite it by executing this script
if (-f FILENAME)
{
    die "Baseline file ".FILENAME." exists.\n";
}

# Open the baseline file for writing
open (my $BASELINE, '>', FILENAME) || die "file could not be opened";

# Execute the nmap command
system($nmap_command_line);

# Parse the nmap.xml file setup the array of all hosts.
$np->parsefile('nmap.xml');
my @host_array = $np->all_hosts();
```

```
# Step through each host array entry and set values.
# Set items that are null/empty to "UNDEFINED".
foreach my $host (@host_array)
{
    if (defined $host->hostname())
    {
        if($host->hostname() eq '0')
        {
            $my_hostname = "UNDEFINED";
        }
        else
        {
            $my_hostname = $host->hostname();
        }
    }
    else
    {
        $my_hostname = $host->ipv4_addr();
    }

    if (defined $host->ipv4_addr())
    {
        $my_ipv4_addr = $host->ipv4_addr();
    }
    else
    {
        $my_ipv4_addr = 'UNDEFINED';
    }

    if (defined $host->mac_addr())
    {
        $my_mac_addr = $host->mac_addr();
    }
    else
    {
        $my_mac_addr = "UNDEFINED";
    }

    if (defined $host->mac_vendor())
    {
        $my_mac_vendor = $host->mac_vendor();
    }
    else
    {
        $my_mac_vendor = "UNDEFINED";
    }

    # Write the entry to the baseline file IP Address, MAC Address,
```

```
    # MAC Vendor String, and Hostname
    print $BASELINE
$my_ipv4_addr.",".$my_mac_addr.",".$my_mac_vendor.",".$my_hostname."\n
";
}
```

Perl script: scan.pl

```
#!/usr/bin/perl
# scan.pl
# Written by Kenton Groombridge
# THIS SOFTWARE COMES WITH ABSOLUTELY NO WARRANTY! USE AT
# YOUR OWN RISK!
use strict;
use warnings;
use Nmap::Parser;

# This is the filename of the baseline file.
use constant BASENAME => "baseline.csv";

# This is the filename created by future scans which will be
# compared against the baseline file.
use constant SUSPICIOUS => "suspicious.csv";

# Option to use Nmap scan
use constant NMAP => 1;
# Make this value the network address of the network to scan.
use constant NETWORK => "192.168.0.0/24";

# This is how often to execute a scan.  Make small enough to
# provide feedback quickly, but not so small to burden the
# network with constant scanning.
use constant INTERVAL => 60;

# Optionally use SSH to access the DHCP device and pull the
# DHCP binding information to compare against the baseline.
# If enabled, change the DHCP server address to that of the
# device responsible for DHCP.
use constant DHCP => 1;
use constant DHCP_SERVER => "192.168.0.11";
use constant DHCPNAME => "dhcp.txt";

# Optionally use SSH to glean the ARP cache usually off the
# default router
use constant ARP => 1;
use constant DEFAULT_ROUTER => "192.168.0.11";
use constant ARPNAME => "arp.txt";

# Optionally use tcpdump to capture all the ARP replies
use constant PASSIVE => 1;
use constant TCPNAME => "tcpdump.txt";

# Optionally use snmptraps, must configure snmptrapd to place
# log file in script directory.
use constant TRAPS => 1;
use constant TRAPNAME => "snmptrapd.log";

# Set up variables and other items required for the script
```

```
my $my_hostname;
my $my_ipv4_addr;
my $my_mac_addr;
my $my_mac_vendor;
my $np = new Nmap::Parser;

# Setup the Nmap command line with appropriate network address defined
# above.  Send default output to /dev/null as it isn't needed
my $nmap_command_line = ("nmap -sn -oX nmap.xml ".NETWORK." >
/dev/null");

# Setup the dhcp command line with address of dhcp server.
my $dhcp_command_line = ("ssh ".DHCP_SERVER." 'cat
/tmp/var/lib/misc/dnsmasq.leases' > ".DHCPNAME);

# Setup the arp command line with default router address.
my $arp_command_line = ("ssh ".DEFAULT_ROUTER." 'arp' > ".ARPNAME);

# Setup the tcpdump command line to save output to tcpdump.txt,
# send stderr to /dev/null.
my $tcpdump_command_line = ("stdbuf -o0 tcpdump -tn arp and
ether[20:2] = 2 > ".TCPNAME." 2> /dev/null");

# Setup the command line to truncate tcpdump.txt.
my $trunctcp_command_line = ("cat /dev/null > ".TCPNAME);

# Setup the command line to truncate the snmptrapd file.
my $trunctrap_command_line = ("cat /dev/null > ".TRAPNAME);

# Run this outside loop or it will spin up endless tcpdump listeners
if (PASSIVE)
{
    # The passive listener isn't a command that is run once
    # and quit, it always has to run, so fork it to the
    # background
    my $pid = fork;

    if (!defined $pid)
    {
        die "Cannot fork, error $!";
    }
    elsif ($pid == 0)
    {
        # This is the client process so start passive listener
        print "Starting tcpdump passive listener...\n";
        system($tcpdump_command_line);
        exit 0;
    }
}

# This is set to run forever once started until CTRL+C or killed
while ()
```

Kenton A. Groombridge, kgroombr@comcast.net

```
{
    open (my $SUSPECT, '>', SUSPICIOUS) || die "file could not be
opened";

    if (NMAP)
    {
        # Open the file to save new scan, execute Nmap, parse
        # the xml file, and setup the array of all host information
        # found in this scan.
        system($nmap_command_line);

        # Parse the nmap.xml file setup the array of all hosts.
        $np->parsefile('nmap.xml');
        my @host_array = $np->all_hosts();

        # Step through each host and set local variables.
        foreach my $host (@host_array)
        {
            if (defined $host->ipv4_addr())
            {
                $my_ipv4_addr = $host->ipv4_addr();
            }
            else
            {
                $my_ipv4_addr = 'UNDEFINED';
            }

            if (defined $host->mac_addr())
            {
                $my_mac_addr = $host->mac_addr();
            }
            else
            {
                $my_mac_addr = "UNDEFINED";
            }

            # Setup some local variables to detect hosts not found in
            # the baseline file via the Nmap scan.
            my $ip_match = 0;
            my $mac_match = 0;
            # Append a comma so IP like 192.168.0.1 doesn't match
            # IP 192.168.0.100 as the baseline is a csv file
            my $find_ip = "$my_ipv4_addr,";
            my $find_mac = "$my_mac_addr";

            # Open the baseline file in which to compare new scan.
            open (my $BASELINE, BASENAME) || die "file could not be
opened";

            # Get one line at a time from the baseline file.
            my @entry = <$BASELINE>;
```

```
            # Loop though each entry and compare it to the information
            # found in the new scan.
            for (@entry)
            {

                if ($_ =~ /$find_ip/)
                {
                    # Found an IP match
                    $ip_match = 1;
                }

                if ($_ =~ /$find_mac/i)
                {
                    # Found a MAC match
                    $mac_match = 1;
                }

                close $BASELINE;
            }

            # Save the appropriate message to the suspect file
            # depending on findings.
            if (not $ip_match and not $mac_match)
            {
                print $SUSPECT $find_ip.$find_mac." - NMAP Scan - New
IP and MAC Address\n";
            }
            elsif (not $ip_match)
            {
                print $SUSPECT $find_ip.$find_mac." - NMAP Scan - New
IP Address\n";
            }
            elsif (not $mac_match)
            {
                print $SUSPECT $find_ip.$find_mac." - NMAP Scan - New
MAC Address\n";
            }
        }
    }

    # Do this all again if looking at the arp cache on the router.

    if (ARP)
    {
        system($arp_command_line);

        # Execute the arp command that will ssh into the
        # dhcp server and redirect output to arp.txt.
        open (my $ARP, ARPNAME) || die "file could not be opened";
```

```
        while (my $arpentry = <$ARP>)
        {
            # Setup some local variables to detect hosts not found in
            # the baseline file via the arp cache.

            my $ip_match = 0;
            my $mac_match = 0;

            # Use a regex to extract IP
            (my $find_ip) = $arpentry =~
/(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})/;

            # Append a comma so IP like 192.168.0.1 doesn't match
            # IP 192.168.0.100 as the baseline is a csv file
            $find_ip = "$find_ip,";

            # Use a regex to extract MAC
            (my $find_mac) = $arpentry =~ /((?:[0-9a-f]{2}[:-]){5}[0-
9a-f]{2})/i;

            # Open the baseline file in which to compare new scan.
            open (my $BASELINE, BASENAME) || die "file could not be
opened";

            # Get one line at a time from the baseline file.
            my @entry = <$BASELINE>;

            # Loop though each entry and compare it to the information
            # found in the new scan.
            for (@entry)
            {
                if ($_ =~ /$find_ip/)
                {
                    # Found an IP match
                    $ip_match = 1;
                }

                if ($_ =~ /$find_mac/i)
                {
                    # Found a MAC match
                    $mac_match = 1;
                }
                close $BASELINE;
            }
            # Save the appropriate message to the suspect file
            # depending on findings.
            if (not $ip_match and not $mac_match)
            {
                print $SUSPECT $find_ip.$find_mac." - ARP Cache - New
IP and MAC Address\n";
            }
```

```
            elsif (not $ip_match)
            {
                print $SUSPECT $find_ip.$find_mac." - ARP Cache - New
IP Address\n";
            }
            elsif (not $mac_match)
            {
                print $SUSPECT $find_ip.$find_mac." - ARP Cache - New
MAC Address\n";
            }
        }
    }

    # Do this again if looking at the dhcp bindings table from router.

    if (DHCP)
    {
        # Execute the dhcp command that will ssh into the
        # dhcp server and redirect output to dhcp.txt.
        system($dhcp_command_line);

        open (my $DHCP, DHCPNAME) || die "file could not be opened";

        # Read from the retrieve dhcp table one line at a time
        while (my $dhcpentry = <$DHCP>)
        {
            # Setup some local variables to detect hosts not found in
            # the baseline file via the dhcp table.

            my $ip_match = 0;
            my $mac_match = 0;

            # Use a regex to extract IP
            (my $find_ip) = $dhcpentry =~
/(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})/;

            # Append a comma so IP like 192.168.0.1 doesn't match
            # IP 192.168.0.100 as the baseline is a csv file
            $find_ip = "$find_ip,";

            # Use a regex to extract MAC
            (my $find_mac) = $dhcpentry =~ /((?:[0-9a-f]{2}[:-]){5}[0-
9a-f]{2})/i;

            # Open the baseline file in which to compare new scan.
            open (my $BASELINE, BASENAME) || die "file could not be
opened";

            # Get one line at a time from the baseline file.
            my @entry = <$BASELINE>;
```

```
            # Loop though each entry and compare it to the information
            # found in the new scan.
            for (@entry)
            {
                if ($_ =~ /$find_ip/)
                {
                    # Found an IP match
                    $ip_match = 1;
                }

                if ($_ =~ /$find_mac/i)
                {
                    # Found a MAC match
                    $mac_match = 1;
                }
                close $BASELINE;
            }
            # Save the appropriate message to the suspect file
            # depending on findings.
            if (not $ip_match and not $mac_match)
            {
                print $SUSPECT $find_ip.$find_mac." - DHCP Table - New
IP and MAC Address\n";
            }
            elsif (not $ip_match)
            {
                print $SUSPECT $find_ip.$find_mac." - DHCP Table - New
IP Address\n";
            }
            elsif (not $mac_match)
            {
                print $SUSPECT $find_ip.$find_mac." - DHCP Table - New
MAC Address\n";
            }
        }
    }

    # Do this all again if looking at the tcpdump output.

    if (PASSIVE)
    {
        open (my $TCP, TCPNAME) || die "file could not be opened";

        # Read from the retrieve dhcp table one line at a time
        while (my $tcpentry = <$TCP>)
        {
            # Setup some local variables to detect hosts not found in
            # the baseline file via the dhcp table.

            my $ip_match = 0;
            my $mac_match = 0;
```

```
            # Use a regex to extract IP
            (my $find_ip) = $tcpentry =~
/(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})/;

            # Append a comma so IP like 192.168.0.1 doesn't match
            # IP 192.168.0.100 as the baseline is a csv file
            $find_ip = "$find_ip,";

            # Use a regex to extract MAC
            (my $find_mac) = $tcpentry =~ /((?:[0-9a-f]{2}[:-]){5}[0-
9a-f]{2})/i;

            # Open the baseline file in which to compare new scan.
            open (my $BASELINE, BASENAME) || die "file could not be
opened";

            # Get one line at a time from the baseline file.
            my @entry = <$BASELINE>;

            # Loop though each entry and compare it to the information
            # found in the new scan.
            for (@entry)
            {
                if ($_ =~ /$find_ip/)
                {
                    # Found an IP match
                    $ip_match = 1;
                }

                if ($_ =~ /$find_mac/i)
                {
                    # Found a MAC match
                    $mac_match = 1;
                }
                close $BASELINE;
            }
            # Save the appropriate message to the suspect file
            # depending on findings.
            if (not $ip_match and not $mac_match)
            {
                print $SUSPECT $find_ip.$find_mac." - PASSIVE - New IP
and MAC Address\n";
            }
            elsif (not $ip_match)
            {
                print $SUSPECT $find_ip.$find_mac." - PASSIVE - New IP
Address\n";
            }
            elsif (not $mac_match)
            {
```

```
                print $SUSPECT $find_ip.$find_mac." - PASSIVE - New
MAC Address\n";
            }
        }
        # Execute the command to truncate the tcpdump file so
        # we start fresh with every loop.
        system($trunctcp_command_line);
    }

    if (TRAPS)
    {
        open (my $TRAP, TRAPNAME) || die "file could not be opened";

        # Declare here so its scope is available in all blocks.
        my $find_ip;

        # Read from the retrieve dhcp table one line at a time
        while (my $trapentry = <$TRAP>)
        {
            # Setup some local variables to detect hosts not found in
            # the baseline file via the dhcp table.

            my $mac_match = 0;

            # Because snmp messages are multi line need to ensure
            # the right line to extract content. Only interested
            # in the line with word "TRAP" and Hex-STRING.

            if ($trapentry =~ "TRAP")
            {
                # Use a regex to extract the first IP it finds.
                # Unlike other checks, don't search on IP, but need to
                # use it in the message so the switch where the device
                # was inserted/removed can be found.
                ($find_ip) = $trapentry =~
/(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})/;
            }
            # If it is the Hex-STRING line, it is the last line
            # so print message.
            elsif ($trapentry =~ "Hex-STRING")
            {
                # Use a regex to extract Hex string
                (my $find_hex) = $trapentry =~ /((?:[0-9a-f]{2}[\
]){11}[0-9a-f]{2})/i;

                # Extract first byte from hexstring to determine event
                my @hex_bytes = split ' ', $find_hex;
                my $snmp_msg = $hex_bytes[0];

                # Put together the MAC address with colons so that it
                # is the same format as the baseline
```

```
                my $find_mac =
"$hex_bytes[3]:$hex_bytes[4]:$hex_bytes[5]:$hex_bytes[6]:$hex_bytes[7]
:$hex_bytes[8]";

                # Open the baseline file in which to compare new scan.
                open (my $BASELINE, BASENAME) || die "file could not
be opened";

                # Get one line at a time from the baseline file.
                my @entry = <$BASELINE>;

                # Loop though each entry and compare it to the
                # information found in the new scan.
                for (@entry)
                {
                    if ($_ =~ /$find_mac/i)
                    {
                        # Found a MAC match
                        $mac_match = 1;
                    }
                    close $BASELINE;
                }
                # Save the appropriate message to the suspect file
                # depending on findings.
                if (not $mac_match and ($snmp_msg == "01"))
                {
                    print $SUSPECT $find_mac." - TRAP - New MAC
Learned at Switch IP $find_ip\n";
                }
                elsif (not $mac_match and ($snmp_msg == "02"))
                {
                    print $SUSPECT $find_mac." - TRAP - New MAC
Flushed at Switch IP $find_ip\n";
                }
            }
        }
        # Execute the command to truncate the snmptrapd log file so
        # we start fresh with every loop.
        system($trunctrap_command_line);
    }


    close $SUSPECT;

    # If the suspicious.csv file is non-empty, send message to admin
    if (-s SUSPICIOUS)
    {
        # Here is where to put method to send message to notify admin
        # along with the file or contents of suspicious.csv
        # Mail, popup, whatever or print contents like this:
        open(my $SUSPECT, SUSPICIOUS) or die "file could not be
opened";
```

Kenton A. Groombridge, kgroombr@comcast.net

```
        while (my $entry = <$SUSPECT>)
        {
            print "$entry";
        }
        print "-----------------------------------------------------
----------\n";
        close $SUSPECT;
    }

    # Delay the interval time until next scan.
    sleep INTERVAL;
}
```


Kenton A. Groombridge, kgroombr@comcast.net