

# SSWR

Scripted Security for Wireless Routers



Written by Douglas Berdeaux  
[Douglas@WeakNetLabs.com](mailto:Douglas@WeakNetLabs.com)

## Syntax

All code, output, and packet captures are listed and in tables. Anything in `monospace` font is an actual command. The *ABSTRACT* explains a lot of acronyms used and gets the reader started off by introducing a few wireless networking concepts. This paper is in two parts, essentially, with an Attack/Defense section within *Part 1: My First Attempt* to get the reader ready for the second part. The *Finish and Final Thoughts* section is simply food for thought and all references can be found at the end of the paper.

## Sections

- **Abstract**
  - **WEP (the quick introduction)**
  - **ARP/MAC**
  - **Initialization Vectors**
  - **My Methodology**
- **Part 1. My First Attempt**
  - **Offensive Attacking**
  - **Defensive Attacking**
- **Part 2. My Embedded Attempt**
- **Finish and Final Thoughts**
- **References**

## ABSTRACT

The first implementation<sup>1</sup> of this type of wireless security, that I have tested, was with Catchme-NG! - WEPGuard. This required a separate machine that would attack a wireless attacker. The separate device would sniff packets on a wired connection, and if more than 20 ARP packets from one single source were detected in 1 second, the system would respond by creating de-authentication packets and sending them to the attacking station via a non-associated Monitor-mode enabled radio.

### *WEP(the quick introduction)*

During an encrypted transaction between a station and an AP using the RC4 WEP encryption, the FCS, or frame check sequence, is added to the tail of the data packet and is made up of the following elements: IV, Key ID, Encrypted Message, and ICV. The process goes as follows and I will be explaining the acronyms in summary below.

- i. The *IV* and *WEP Key* are setup by the *KSA*<sup>2</sup>
- ii. The *PGRA*<sup>3</sup> creates a Keystream
- iii. A cyclic redundancy check (*ICV*)<sup>4</sup> is made and the result is appended to the *Message*.
- iv. The *Message*, *ICV*, and the *Keystream* are XORed together creating an encrypted message.
- v. The *Encrypted Message* is appended to the *Key ID*<sup>5</sup>, which is then appended to the *IV*.
- vi. Lastly, the *ICV* is appended to the *Encrypted Message*.

*Listing 0: The WEP encryption method.*

This may seem like a lot to take in, for someone unfamiliar with the whole concept, but it is actually fairly simple to understand. I have italicized the key points and added footnotes for the acronyms in which I don't explain further.

My implementation of SSWR does not, in any way, make WEP or its algorithms stronger. The encryption has been hacked over and over again, and new methods have been published to crack the key in 15~30 seconds, depending on the length.

### *ARP / MAC*

ARP, or Address Resolution Protocol, is critical networking protocol to tell an AP<sup>6</sup> who has a specific IP (Internet Protocol) address. The destination can often be the address `ff:ff:ff:ff:ff:ff` which is the broadcast address. The broadcast address, sends packets to all ports on the LAN/WLAN in search (“*who has*”) of a particular IP address. IP addresses are associated with a client or the AP's MAC (Media Access Controller) address. A router can

<sup>1</sup> Videos and Weblog post links can be found in the references section.

<sup>2</sup> Key Scheduling Algorithm – sets up the Initialization vector and the WEP key.

<sup>3</sup> Pseudo Random Generation Algorithm – creates the keystream from the KSA's setup.

<sup>4</sup> Checksum of the data.

<sup>5</sup> AP's can use multiple WEP keys, each having its own ID. This ID is used by the receiver to decrypt the packets with the proper WEP key. The WEP key is not sent over the air, at all.

<sup>6</sup> Access Point, or Wireless Router.

store tables of MAC address to IP address, similar to an associative array in computer science. If the router does not know a particular IP address, and it is requested by a station either wired or wireless, it will send a broadcast in search of it. An ARP request from a router is broadcasted to all ports on the router, and all ports on all subsequent switching, or level two, devices. It is also sent to all wireless devices as well. This is why this attack is so easy to detect. A normal operation of a properly-designed LAN or WLAN would not ARP more than twenty times per second. Poorly implemented LANs and WLANs could possibly, though, if a loop existed somewhere causing a *broadcast storm*.

No two, or more, radios have the same MAC address. Here is an example MAC address:

```
00:11:22:33:44:55
```

This is, obviously, a fake MAC. The first three sets, or bytes, of the MAC address are manufacturer specific. This means that “00:11:22,” in the above example MAC, would belong to whoever manufactured the physical device. These address bytes, called the OUI or Organizationally Unique Identifier, are pre-assigned to the manufacturer, before they are made in their facilities, and are hard-programmed, or *burned-in*, to the device.

### *Initialization Vectors*

ARP packet replay is the fastest way to generate lots of traffic on a wireless network. Once you have captured a single packet, you can replay it at the router at high speeds, and most wireless routers are forced to respond to the replay by sending acknowledgment (ACK) packets. The ARP packets include an initialization vector, or IV. This IV is a plain-text 24bit section of the data packet which Aircrack-ng uses to statistically guess the WEP key used for encryption. The reason why you can replay the IV is due to the small namespace in which IV's live, which is around 16 million or so according to the Offensive Security Wireless Professional course materials.

The IV is 24bit's making the encryption strength that much smaller. A 64bit WEP key is actually 40bit, and a 128bit WEP key is only 104bits.

### *My Methodology*

This method, in which I propose, is for those who have a Linux, or UNIX, based wireless AP in their Lab, corporation, store, office, or home in which they are forced to use WEP encryption due to hardware or technology restrictions.

This paper is intended for a slightly advanced audience in the subject of wireless transmission protocols, Wireless Equivalency Privacy and the methods used to crack it. I will try to explain as much as possible, while being brief.



Part 1.  
**My First Attempt.**

In the beginning, I created a defense system out of a machine and tools which, up until then, I used in an offensive fashion. This was a device, desktop or laptop, which sat within close vicinity of the Service Set using WEP as its primary security. I used a laptop which had a wireless adapter in monitor mode with a driver suitable enough to enable the device to inject packets into the Service Set. To inject the packets, I used Aireplay-ng of the Aircrack Suite of wireless penetration testing tools. The Ethernet connection of the laptop was set to sniff all traffic of the Address Resolution Protocol.<sup>7</sup> For this, I used TCPDump with these settings:

```
root@wepguard~:#tcpdump -lnt -i eth0 > output.txt
```

*Code Listing 0: TCPDump running on the laptop*

The `-l` argument tells TCPDump to not buffer the packets and release the data until the packet buffer is full, but to display them as they are detected. The `-e` argument, tells TCPDump to display the MAC addresses, from the link level headers of the packets, of the source and destination. The `-n` and `-t` arguments tell TCPDump to not display the hostnames and timestamps of the packets, respectively. Lastly, I specify the protocol to filter; ARP.

This created a file from the Unix shell pipe “>” called `output.txt`. Then, I wrote a simple Perl script which read the file every 1 or 2 seconds using a slurp method, for speed's sake. If the file contained more than 20 lines, I parsed the text and pulled out the MAC of the ARP-replaying MAC address. I then logged this activity, and, knowing my BSSID<sup>8</sup> and ESSID, constructed a packet to de-authenticate the ARP-replaying MAC using Aireplay-ng.

This method worked fine, but required a dedicated machine, electricity, generated heat, and just seemed impractical and inefficient. This method only stops the attacker for a few seconds, causing a mere inconvenience rather than really stopping him or her from the full attack. The WEP key can still be guessed after gathering enough packets, but the attack simply takes a much longer time.

---

<sup>7</sup> See the ABSTRACT for more details.

<sup>8</sup> Basic Service Set Identifier, or MAC address of the AP.

## Offensive Attacking

### *You are the Attacker.*

If one were to set the WEP-Guard enabled laptop to constantly DDoS<sup>9</sup> the attacker, he or she could simply spoof the MAC address<sup>10</sup> and start the attack again. This causes the radio of the WEP-Guard enabled laptop to be in use, practically doing nothing, until it sees new ARPs generated by the new, spoofed MAC. Here is how an attacker spoofs, or changes his/her MAC address:

```
macchanger -m 00:11:22:33:44:55 wlan0
or
ifconfig wlan0 hw ether 00:11:22:33:44:55
```

*Code Listing 1: Spoofing the MAC Address of the Attacker's Radio.*

Here is the TCPDump output from the wireless device in monitor mode. This is one packet from the WEP-Guard Enabled Laptop, received by the attacker. This packet stopped the attacker's ARP packet replay attack dead in its tracks.

```
476288338us tsft 1.0 Mb/s 2427 MHz 11b -39dB signal antenna 1 [bit 14] 314us
BSSID:00:1f:90:e9:50:c4 DA:00:1f:90:e9:50:c4 SA:00:11:22:33:44:55
DeAuthentication: Class 3 frame received from nonassociated station
```

*Packet Listing 0: The WEP-Guard packet that stops the ARP packet replaying attacker.*

Here you can see the Source address is 00:11:22:33:44:55. As the attacker, if we change our MAC to 00:11:22:33:44:66 and restart the attack, it will continue to generate ARP reply's from the AP, after another fake association/authentication and detecting a real ARP packet. The WEP-Guard can stop sending the de-authentication packets, and change its MAC to 00:11:22:33:44:66, then start the guard process over again.

The script, actually, only sends three de-authentication packets to the attacker. The attacker could send keep-alive packets at the router like so

```
aireplay-ng -l 999 -q 10 -a 00:1f:90:e9:50:c4 -h 00:11:22:33:44:55 -e
WeakNetLabs mon0
```

*Code Listing 2: Sending "keep alive" packets after authentication and association has occurred.*

But, alas, this only speeds up the WEPGuard's final attempt to stop you, the attacker. This is to disable the radio until a system administrator can come by or remote into the router to re-enable it.

If the WEP key is only a 40bit key, you may be able to get enough initialization vectors by ARP amplification and guessing the IP of the router and any wireless client. Using `packetforge-ng` the attacker could create a fake ARP request using a captured PRGA and

<sup>9</sup> Dedicated Denial of Service.

<sup>10</sup> This does not change the hard-coded MAC bytes of the hardware, but merely creates a software mask of the new MAC. The old MAC is returned once the adapter, or radio, has been turned off and back on again, or ejected and re-inserted into the system.

use the IP of the router and the wireless client to amplify, or cause up to times the packet speed from the router. Within those few seconds you may be able to get around nine thousand initialization vectors, or just enough to crack a 40bit key. Guessing the IP's can be hard, but if you see a default ESSID, like "**Linksys**," you may want to try the default IP range and router addresses in your `packetforge-ng` command!

### *Penetrating from a different source*

If the ultimate goal is to find the WEP key, and the attacker bypasses the wireless attack vector and can crack the perimeter and gain network to the wired LAN, he or she could scan the network in search of a MAC that matches the vendor bytes of the BSSID, or Wireless LAN MAC. If, after a quick port scan and remote exploit or brute force, the attacker gains access to the wireless device via poorly implemented telnet or SSH, he or she could view the WEP key in plain text by issuing an `iwconfig` command. Here is the output from my router in the lab.

```
ath0      IEEE 802.11g  ESSID:"WeakNetLabs"  
Mode:Master  Frequency:2.427 GHz  Access Point: 00:1F:90:E9:50:C4  
Bit Rate:0 kb/s  Tx-Power=20 dBm  Sensitivity=1/1  
Retry:off  RTS thr:off  Fragment thr:off  
Encryption key:1337-CAFE-69  Security mode:restricted  
Power Management:off  
Link Quality=0/70  Signal level=-96 dBm  Noise level=-96 dBm  
Rx invalid nwid:7470  Rx invalid crypt:0  Rx invalid frag:0  
Tx excessive retries:0  Invalid misc:0  Missed beacon:0
```

*Output Listing 0: The wireless radio from my wireless router.*

They attacker could then change the key by issuing this command:

```
iwconfig ath0 key 1337cafe22
```

*Code Listing 3: Changing the WEP key on the router's main master mode enabled device.*

If you have full terminal access to the router as root, or admin, you will find that the configuration and system administration commands are quite similar to those of a radio, or wireless adapter, in Managed mode on a laptop.

## Defensive Attacking

*You are the defense.*

There is no method, that I know of, to drop packets from a wireless device by specifying the MAC address. One can set up `iptables` to drop all packets from a specific MAC address on, say, wlan0 from the MAC 00:11:22:33:44:55, but this will not stop the WEPGuard from guarding it's Service Set. The packets generated and sent by Aireplay-ng are not ignored by the wlan0 device and the ARP replay attack is stopped, dead in its tracks!

This is because of Wireless Modes. *Master mode* is when the device is acting like an AP. *Managed mode* is when you are acting like a client. And finally, *Monitor mode*, which isn't really a wireless "mode" so to speak, is when you are in promiscuous mode accepting all packets and injecting. This, unfortunately, is why the attack mentioned above fails when the attacker tries to drop all packets from the WEPGuard enabled system. You need to be completely promiscuous and accept the WEPGuards de-authentication packets to simply replay ARP packet in your attack.

You could set up a MAC white list, which is simply a table that drops all packets from MAC addresses that are not in the table. This can, once again, be foiled by the attacker though, as he or she can spoof their MAC address to that of a legitimate client. This would be similar to creating `iptables` rules that DROP or REJECT packets not from the MACs specified in the router's OS.

ARP amplification can be avoided, slightly, by changing your default IP ranges in your wireless LAN and of the AP itself. Don't leave it 192.168.1.1 and your wireless clients 192.168.1.100+ because this could be easily guessed by an attacker, especially if you leave your ESSID "Linksys," "dlink" or "NETGEAR!" Another thing you may want to consider is using a 104bit WEP key rather than 64bit. This lessens the possibility of the attacker cracking the key using ARP amplification.

Part 2.  
**My Embedded Attempt.**

After given the idea from a few colleagues and mulling it over for a few weeks, I decided to try to implement this directly on my wireless router. The response time, needless to say, was incredibly faster. My DDWRT is version 24, and I have it installed on my **MI424-WR rev D**, Actiontec router. This version came with `ash`, `tcpdump`, and the Aircrack Suite. I coded the check script in the `ash` shell scripting language as the router did not have Perl. I created a VAP<sup>11</sup> `ath1337` by issuing a simple `wlanconfig` command like so.

```
wlanconfig ath1337 create wlandev wifi0 wlanmode monitor
```

*Code Listing 4: creating a VAP from ath0 on the router.*

The VAP, I found, would not inject the de-authentication packets during the real live tests. This became a small problem which I resolved by turning the radio off before injecting with `ath1337`. The main “Master” mode device `ath0` would not inject packets into its own Service Set, as it would return the error:

```
ioctl(SIOCSIWMODE) failed: Invalid argument

ARP linktype is set to 1 (Ethernet) - expected ARPHRD_IEEE80211,
ARPHRD_IEEE80211_FULL or ARPHRD_IEEE80211_PRISM instead. Make
sure RFMON is enabled: run 'airmon-ng start ath0 <#>'
Sysfs injection support was not found either.
```

*Output Listing 1: The error returned while trying to inject packets with a Master Mode device.*

The application itself is pretty straight forward. It consists of three shell scripts and one output file. The first shell script is the `installer.sh` script. This script creates the directory `/var/log/wepgrd` for logging purposes, checks to see which driver you are using, makes sure you have the proper dependencies, and changes the `wepgrd.sh` script accordingly with a single `sed` in-place substitution. The next, `wepgrd.sh` is the actual application. This spawns a new process, `linechk.sh`, which starts `tcpdump` with the correct arguments and dumps the `STDOUT` to a file `output.txt`. `Wepgrd.sh` then checks the file every second, and overwrites all the data in it. I had a small problem with this script when I began writing this. I would start the script and `tcpdump` would start, but wouldn't overwrite the contents of the file properly. After I killed the script `wepgrd.sh`, all of the contents that should have been dumped to `output.txt` would suddenly drop into the file! `TCPDump`, unknowingly to me, buffers all output before dumping it to `STDOUT`. The `-l` option makes `TCPDump` act in a line buffered mode, which will display each line as it comes.

With this issue out of the way, I create three variables `$DEVICE`, `$ATTKR` and `$PCKTS`. The first is simply an `Awk/Sed/Grep` of the output from `iwconfig`. The second is same, but of the MAC address source field the MAC to ultimately attack with de-authentication packets, from the file `output.txt`. The second is the number of lines in which `$ATTKR` exists. This uses word count as `wc -l`, and if it is above twenty per read, which is every second, then the router suspects that it is under attack and begins its own attack against the attacker `$ATTKR`. If `$PCKTS` is less than twenty, than the script makes `$ATTKR`

---

<sup>11</sup> VAP stands for Virtual Access Point and is actually a pseudo device created directly from a physical device which remains in Managed mode. This is the trend of wireless drivers and is now implemented on `rtl`, `atheros`, and `BCM`.

equal "" (nothing), and sleeps for one second before reading the file once again in a loop that will go on until a kill signal is received, and the size of \$m is greater than three.

The attack the router performs on the attacking station is a simple Aireplay-ng command like so:

```
aireplay-ng -0 3 -a $BSSID -c $ATTKR $DEVICE &
```

*Code Listing 5: the attack line from the script. This performs a DoS attack against an attacker.*

This is just enough to stop the attacker. Since the command is sent to the background by the shell, the script can continue immediately. The radio is disabled right before the attack takes place and is re-enabled afterwards.

After attacking the attacker, \$m is incremented by one using `let`. An `if` statement then checks to see if \$m is greater than three. If so, the radio is completely turned off until an administrator remotes into the device via the wired connection to fix the issue.

## Finish and Final Thoughts

I will conclude this paper by stating that the absolute, best protection against this attack, would be to take apart each packet that comes into the sniffing ethernet device. Then use libpcap<sup>12</sup> to create a *struct* or take-apart-analysis of the packet. Then, put that IV into a safe spot that cannot be poisoned by any means. Once done, count that amount of times the IV has been using within 1 second or so. If more than, say, 30 then you can start the attack against the attacker, with the rest of the *struct* from libpcap (BSSID, ESSID, Client, etc).

This method would be more suited being off-loaded onto a system with a faster CPU and more memory than that of today's average consumer-based router. Almost similar to that of today's practices with a high powered Snort based IDS, or Intrusion Detection System. Alas, this will take far too much time and make the network seem to drag or become slow, especially to those using the internet for multimedia, or the local LAN for streaming media.

Another potential aspect would be to set up a `sendmail` client on the router and email the system administrator when the radio goes down via the DS or wired network connection.

Where this method and code fails, is when you have an instance where at attacker simply needs to perform a DoS attack on a wireless router, as the radio turns off after 4 attacks.

SSWR can, and most likely will, be expanded upon to attack Stations who connect to a Rogue Access point. This can be achieved by storing all known BSSID's and then sending de-authentication packets to clients who connect to a BSSID not within the array. Initializing the radio and disrupting normal operation traffic would be unwise. This creates the illusion that something is wrong with the Rogue AP and that it is not useable. Once again, I would suggest having another router nearby your Service Sets to perform this action.

---

12 Libpcap is a C library for analyzing, and forging RAW packet frames or packet captures.

## References

### SSWR

Aircrack Suite

<http://www.aircrack-ng.org/>

Weblog post of Catchme-NG WEPGuard version 1

<http://trevelyn.com/?p=141>

YouTube Post of Catchme-NG WEPGuard version 1 and brainstorm

[http://www.youtube.com/watch?v=L4W\\_Yp87Fc](http://www.youtube.com/watch?v=L4W_Yp87Fc)

Offensive Security OSWP

<http://www.offensive-security.com/online-information-security-training/backtrack-wifu/>

DDWRT

<http://www.dd-wrt.com/site/index>

OPENWRT

<http://openwrt.org/>

TCPDUMP

<http://www.tcpdump.org/>

IPTABLES

<http://www.netfilter.org/projects/iptables/index.html>

Actiontec

<http://www.actiontec.com/index.php>