



How to protect networks with wireless devices (802.xx/Zigbee/etc...)

This presentation is about a specific implementation of a methodology. The point here is to promote thinking about how to use this kind of thinking to secure all types of wireless networks on all types of platforms. Once an attacker is on the LAN all of the egress point protections are averted.

By: Timber Wolfe (lonegray@gmail.com)

romanHunter

(ROuter huMAN Hunter)

- romanHunter is a system requiring a Netgear **WG602Vx** wireless access point and a device that can run python scripts

- Will collect a set of MAC addresses for use as a data stream for MAC black listing
- Think outside of the box, this methodology can be applied to any wireless router

Reverse Engineering the Router (WG602v4 Screen Shots)

- Stations descriptions how this router works
 - No Stations Listed
 - Associations (wireless attempted connections)
 - Authorizations (wireless password accepted)
- HTTPScoop (Reverse Engineering the Conversation)
 - URLs of Interest
 - <http://192.168.0.227/cgi-bin/stalist.html> (where the MAC addresses come from)
 - <http://192.168.0.227/cgi-bin/security.html> (Where the security settings are performed)

WG602v4 Defaults and Notes

- Default IP: <http://192.168.0.227>
- Default Username: admin
- Default PW: password

No Stations Listed

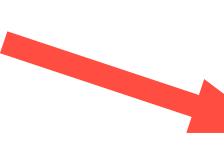
NETGEAR® settings
54 Mbps Wireless Access Point **WG602v4**

Information
Setup
IP Settings
Wireless Settings
Security Settings
Access Control
Management
Change Password
Upgrade Firmware
Restore Factory Default
Station List
Reboot AP
Advanced
Wireless Settings
Wireless Bridging
Web Support
Knowledge Base
Documentation
Logout

Station List

Station ID	MAC Address	Channel	Status
(There is no active connection)			

Refresh



Associations

NETGEAR® settings
54 Mbps Wireless Access Point WG602v4

Information

- [Setup](#)
- [IP Settings](#)
- [Wireless Settings](#)
- [Security Settings](#)
- [Access Control](#)

Management

- [Change Password](#)
- [Upgrade Firmware](#)
- [Restore Factory Default](#)
- [Station List](#)
- [Reboot AP](#)

Advanced

- [Wireless Settings](#)
- [Wireless Bridging](#)

Web Support

- [Knowledge Base](#)
- [Documentation](#)

Station List

Station ID	MAC Address	Channel	Status
1	88:32:9B:7B:A2:D4	11	Associated

[Refresh](#)



Authorizations

NETGEAR® settings
54 Mbps Wireless Access Point **WG602v4**

Information	Station List										
<ul style="list-style-type: none">• Setup• IP Settings• Wireless Settings• Security Settings• Access Control <ul style="list-style-type: none">• Management• Change Password• Upgrade Firmware• Restore Factory Default• Station List• Reboot AP <ul style="list-style-type: none">• Advanced• Wireless Settings• Wireless Bridging <ul style="list-style-type: none">• Web Support• Knowledge Base• Documentation Logout	<table border="1"><thead><tr><th>Station ID</th><th>MAC Address</th><th>Channel</th><th>Status</th></tr></thead><tbody><tr><td>1</td><td>88:32:9B:7B:A2:D4</td><td>11</td><td>Authorized</td></tr></tbody></table> Refresh	Station ID	MAC Address	Channel	Status	1	88:32:9B:7B:A2:D4	11	Authorized		
Station ID	MAC Address	Channel	Status								
1	88:32:9B:7B:A2:D4	11	Authorized								

HTTPScoop Screenshots

Reverse engineering the
router protocol with the client

Viewing the Existing Connections on the Router

The screenshot shows the HTTPScoop interface with two main panes. The left pane displays a list of network connections with details like Time, Method, Request URL, Content Type, Size (KB), Status, and Code. The right pane shows the raw HTTP request and response text.

HTTP Request Detail:

```
<td nowrap align="center"><b>Station ID</b> </td>
<td nowrap align="center"><b>MAC Address</b></td>
<td nowrap align="center"><b>Channel</b></td>
<td nowrap align="center"><b>Status</b></td>
</tr>

<script language="javaScript">
var sta_list;
var assoc_list="";
var autho_list="";
if(autho_list.length < 17)
    sta_list = assoc_list;
else if(assoc_list.length < 17)
    sta_list = autho_list;
else
    sta_list = assoc_list + ' ' + autho_list;
var channel = '11';
var mactext = new Array();
var alltext = new Array();
var macarr = new Array();
```

Two red arrows point to the script block in the Response Text pane. One arrow points to the opening tag <script language="javaScript">, and another arrow points to the closing tag </script>.

Time	Method	Request URL	Content Type	Size (KB)	Status	Code
21:59:12:322	GET	http://192.168.0.227/cgi-bin/stalist.html	text/html	3.90	Done [0.000s]	200
21:59:12:816	GET	http://192.168.0.227/form.css	text/css	0	Done [0.001s]	304
21:59:12:817	GET	http://192.168.0.227/func.js	application/x-javascript	0	Done [0.000s]	304
21:59:12:818	GET	http://192.168.0.227/liteblue.gif	image/gif	0	Done [0.000s]	304
21:59:12:819	GET	http://192.168.0.227/h.stalist.html	text/html	0	Done [0.000s]	304
21:59:12:821	GET	http://192.168.0.227/form.css	text/css	0	Done [0.000s]	304

An Association

- An association is an attempted connection that did not get authorized.
- The red arrow denotes the associated MAC address.

```
3>
40
41 <script language="JavaScript">
42 var sta_list;
43 var assoc_list='assoclist 88:32:9B:7B:A2:D4';
44 var autho_list='';
45 if(autho_list.length < 17)
46   sta_list = assoc_list;
47 else if(assoc_list.length < 17)
48   sta_list = autho_list;
49 else
```



An Authorization

- An authorization is a MAC that will be displayed when that user has entered the correct password.
- The red arrow denotes the authorized MAC address.
- An authorized MAC is also associated.

```
41 <script language="JavaScript">
42 var sta_list;
43 var assoc_list='assoclist 88:32:9B:7B:A2:D4';
44 var autho_list='autho_sta_list 88:32:9B:7B:A2:D4';
45 if(autho_list.length < 17)
46   sta_list = assoc_list;
47 else if(assoc_list.length < 17)
48   sta_list = autho_list;
49 else
```



An Authorization

- An authorization is a MAC that will be displayed when that user has entered the correct password.
- The red arrow denotes the authorized MAC address.
- An authorized MAC is also associated.

```
41 <script language="JavaScript">
42 var sta_list;
43 var assoc_list='assoclist 88:32:9B:7B:A2:D4';
44 var autho_list='autho_sta_list 88:32:9B:7B:A2:D4';
45 if(autho_list.length < 17)
46   sta_list = assoc_list;
47 else if(assoc_list.length < 17)
48   sta_list = autho_list;
49 else
```



Changing the Wireless PW on the Router

- To change the PW on this router several variables must be submitted to the **wpas.html** page.
 - **setobject_security_type=4**
 - **setobject_wpapskPhrase=TheNewPassword**
- If WEP is enabled the variables required must be scooped. This ONLY applies when WPA is in use.
- Also, keep in mind that realm security is used. Each page must include that realm token in each request for it to be accepted.

Roman Hunter Overview

File Structure

- romanHunterv3C.py (Python Script)
- romanHunterv3.txt (Logging File)
- pw_list.txt
(File containing the list of passwords to use)

romanHunterv3.txt Sample

```
374 2013-10-21 19:46:59,549 ERROR found: A8:88:08:88:F9:20
375 2013-10-21 19:46:59,550 ERROR Changing pw from: donttrustgoats
376 to: zebrahunter
377
378 2013-10-22 08:04:37,778 ERROR Error 100, could not open URL
379 2013-10-31 17:04:18,146 ERROR found: 88:32:9B:7B:A2:D4
380 2013-10-31 17:04:18,152 ERROR Changing pw from: zebrahunter
381 to: goatsrevil
382
383 2013-10-31 17:04:42,225 ERROR found: 88:32:9B:7B:A2:D4
384 2013-10-31 17:04:42,226 ERROR Changing pw from: goatsrevil
385 to: willctthese
386
387 2013-10-31 19:07:59,070 ERROR found: 24:77:03:BD:33:80
388 2013-10-31 19:07:59,077 ERROR Changing pw from: willctthese
389 to: wantthatmac1
390
391 2013-11-09 09:53:54,209 ERROR found: 88:32:9B:7B:A2:D4
392 2013-11-09 09:53:54,215 ERROR Changing pw from: wantthatmac1
393 to: easyeasy1
394
395 2013-11-09 09:58:35,937 ERROR found: 88:32:9B:7B:A2:D4
396 2013-11-09 09:58:35,938 ERROR Changing pw from: easyeasy1
397 to: easyeasy2
398
```

pw_list.txt Sample

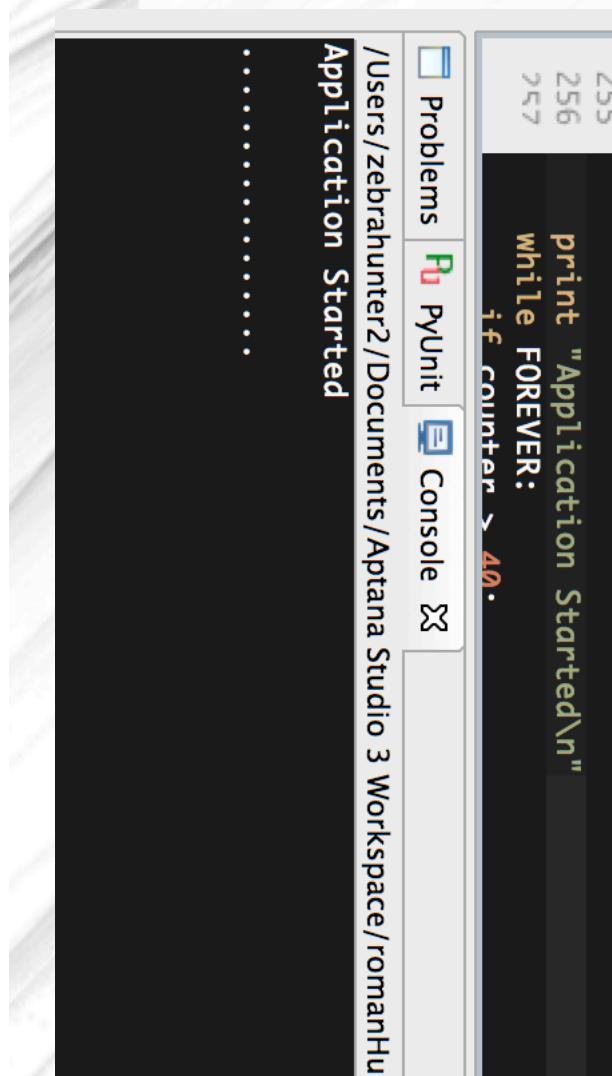
1	zebrahunter
2	goatsrevil
3	willcthese
4	wantthatmac!
5	easyeasy1
6	easyeasy2
7	beesscareme
8	dontrustgoats

romanHunter Screen Shots

The following screenshots illustrate the
romanHunter in action

Idle, illustrating the heartbeat signal

- Note the dots, they illustrate the heartbeat signal. This denotes the script is running and is not monitoring any activity.
- It is critical to know the Python script monitoring one or more honeypots is not locked up.



A screenshot of an IDE interface showing a Python script. The code is as follows:

```
 255 print "Application Started\n"
256
257 while FOREVER:
258     if counter > 40:
259         ....
```

The script prints "Application Started" and then enters an infinite loop. Inside the loop, it checks if the variable `counter` is greater than 40. If it is, the script continues to run, indicated by the ellipsis (...).

The IDE interface includes tabs for Problems, PyUnit, and Console, and shows the file path /Users/zebrahunter2/Documents/Aptana Studio 3 Workspace/romanHur. The status bar at the bottom indicates "Application Started".

The Output of an Association

- The following output is what would be produced when an association is seen by the script.
 - Associations denote 'attempted' connections, if they are also not authorized.



The Output of an Authorization

- Note the line that says 'Hacker MAC'. That is the MAC address of the machine we are after.
- THIS MAC address is the point of this exercise. This information can then be used to block the hackers attempts to access nearby networks and alert CERT/CIRT members. The MAC could also be used to direct the attacker to a honeypot.

```
.....Associated MAC: 88:32:9B:7B:A2:D4.....  
Hacker MAC: 88:32:9B:7B:A2:D4  
.....Associated MAC: 88:32:9B:7B:A2:D4.....
```



Basic Script Logic

- Run Main()
- Call check4ConnectionsR()
- Call changePassword() (if applicable)
- Call generateNewWIFIPW() (if applicable)

Caveats for a Working Environment

- If you are using WEP Security:
 - There does not need to be any additional configuration or connections
- If you are using WPA or anything other than WEP:
 - A device of any kind MUST be connected to the access point, once its in production. This is required for the attacker to disconnect.

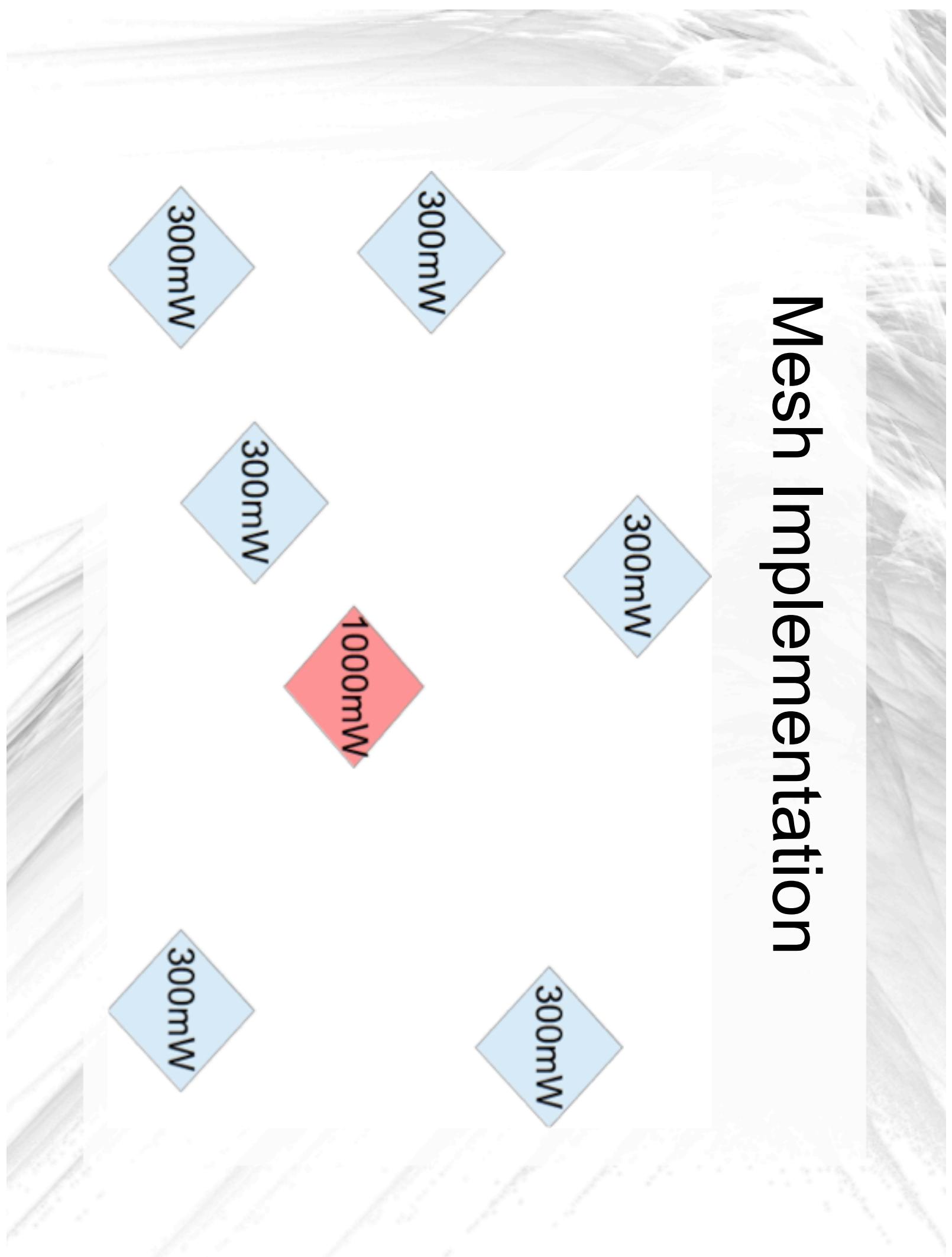
Roman Hunter Screen Shots

- Stations descriptions how this router works
 - No Stations Listed
 - Associations (wireless attempted connections)
 - Authorizations (wireless password accepted)
- Python scripts in action
- Heartbeat signal the *romanHunter* produces

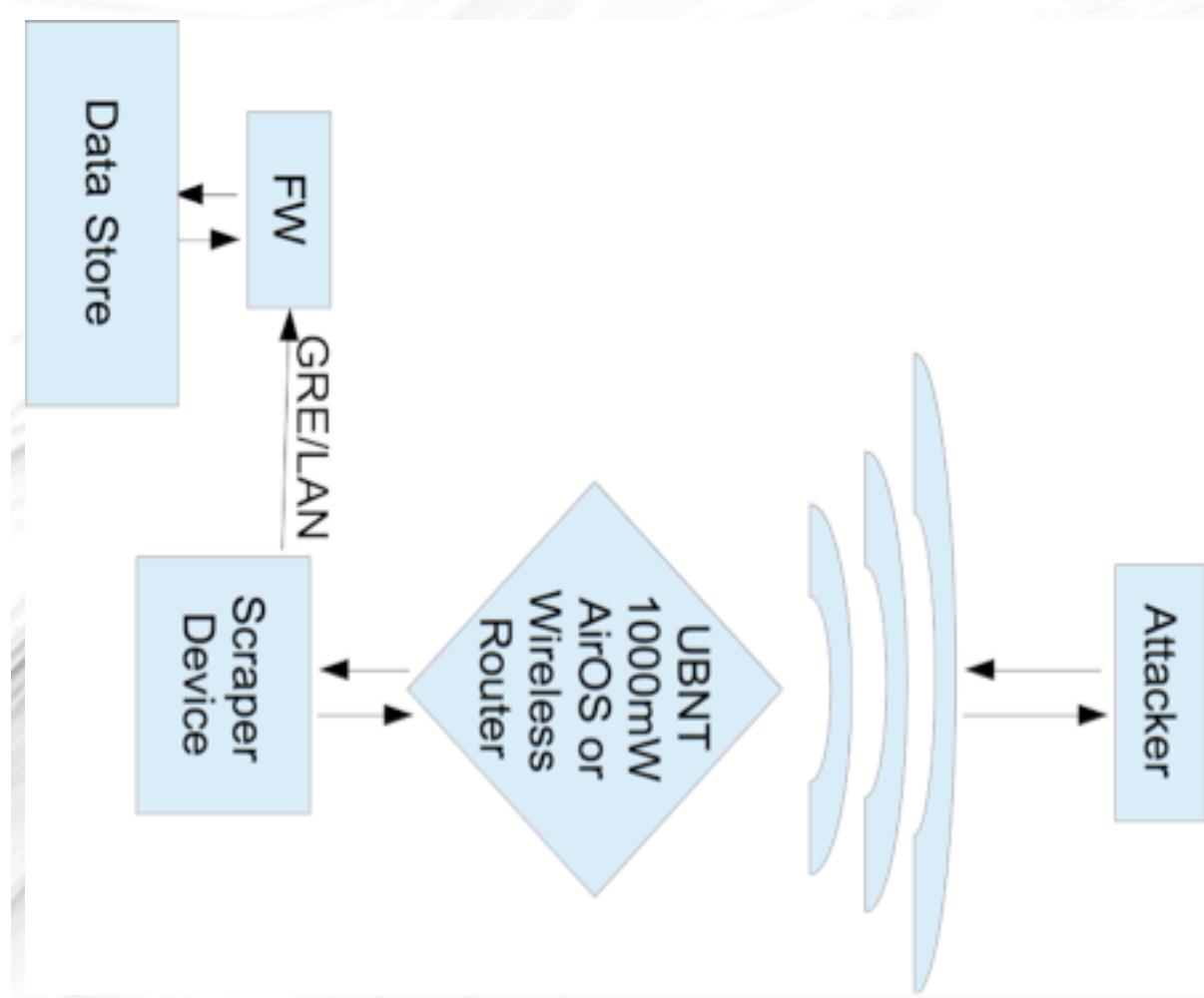
Implementations

- Wireless Mesh
- Adjacent AP to a corporate network
- Standalone AP (pen tests, testing for future office location)
- Can be used with WEP for faster hacker identification (15 minutes to compromise)
- Can be used to illustrate time to compromise with current PW implementations and infrastructure configurations as well as the distance the AP reaches, with zero impact on the customer network(s)

Mesh Implementation



Typical Deployment



Python Script Review

- Functionalized
- Limitations
 - Not enough error checking
 - Should include more abstraction for functionality that does not change between different router versions and types
 - Should include database functionality for storing the MAC addresses
 - Should include a function for uploading to a central repository of MAC's found

Tools Used During Development

- OS X Laptop
- HTTPScoop (Fiddler2 on Windows)
- VM Ware (Fusion 5)
- Windows 7 VM (initial setup of AP)
- Python (for scripting)
(Can use any scripting language)
- Cell phone with wireless to simulate attacker

Tools Used

- **HTTPScoop HTTP Protocol Analyzer.**
 - High Level protocol analyzer usage is easier to comprehend and find instructions versus something like wireshark.
 - OS X (can use: fiddler2 for Windows)

Other Tools for Extending romanHunter

- Scapy (protocol dissector and manipulator)
- MySQL
- Wireshark

