EAPeak - Wireless 802.1X EAP Identification and Foot Printing Tool

Matt Neely and Spencer McIntyre

Synopsis

In this paper we present how to determine the EAP type used by an 802.11 network using 802.1X Enterprise Authentication, and what useful information can be learned from examining an EAP transaction. When attacking an 802.11 network that uses 802.1X Enterprise Authentication, it is key to know what Extensible Authentication Protocol (EAP) type is being used to authenticate the client. The EAP type used by the network will greatly influence which attacks can be successfully launched to gain access to the network. Common EAP types used by wireless networks include PEAP, EAP-TTLS, EAP-TLS, EAP-Fast, and LEAP. In this paper we discuss how to manually determine the EAP type used by a network, and introduce a new tool, EAPeak, that automates this process.

Authors

Name	Revision Title	Date
Matt Neely and Spencer McIntyre	1.0	March 1, 2011

Table of Contents

Background

Wireless security has evolved greatly from the time 802.11 was first introduced. Wired Equivalent Privacy (WEP) was the first attempt at securing 802.11. However, holes were quickly found in WEP, and today WEP is considered a deprecated security protocol.

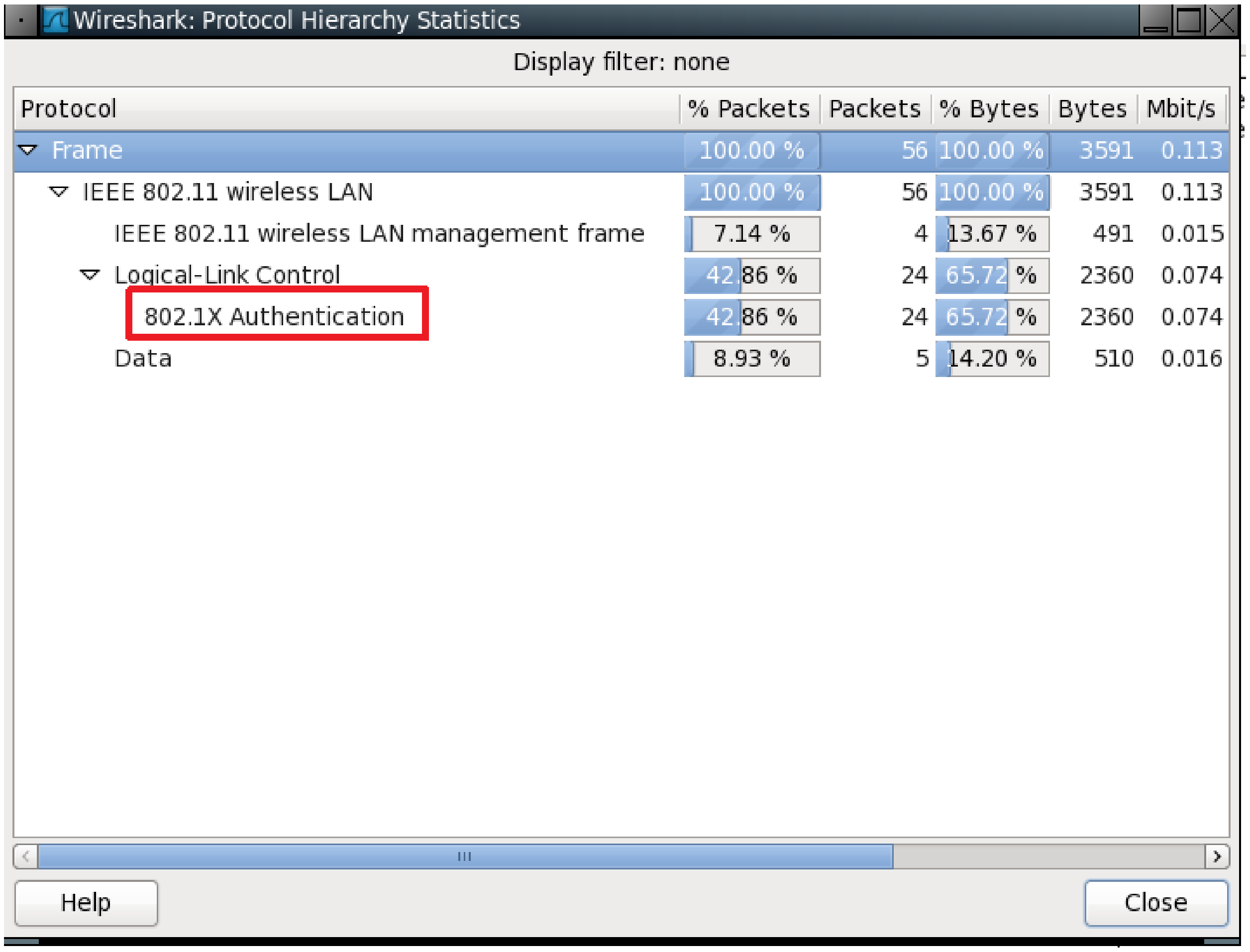
WPA was developed to overcome the weaknesses in WEP. WPA split authentication and encryption into two separate items, and can provide strong encryption and multiple authentication mechanisms. WPA supports two forms of authentication Pre-Shared Key (PSK) and Enterprise. This paper will focus on Enterprise Authentication. WPA encryption types will not be covered.

Enterprise Authentication uses 802.1X as the authentication framework. Unlike other wireless authentication models such as WEP and PSK, Enterprise Authentication can provide per-use or per-system authentication. 802.1X framework uses Extensible Authentication Protocol (EAP) to perform the authentication. Common EAP types used by wireless networks include PEAP, EAP-TTLS, EAP-TLS, EAP-Fast, and LEAP. The EAP types used and how it is configured can greatly impact the security of the wireless network. Therefore, when penetration testing a network that uses Enterprise Authentication it is critical to know the EAP types used to authenticate clients to the network because the EAP types used by the network will greatly influence which attacks can be successfully launched to gain access to the network. For example, if LEAP is being used, a penetration tester may decide to use a tool such as ASLEAP to attack the network. However, if PEAP is being used, Joshua Wright and Brad Antoniewicz's rogue RADIUS server attack would be an appropriate attack. In the past many penetration testers would guess at the EAP type being used or would manually analyze network traffic to determine the EAP type. This paper first will outline how to manually determine the EAP types used by a wireless network. Once the manual process is covered, we will discuss EAPeak, a new tool that automates this process.

Manually Determining the EAP Type

TCPDump and Wireshark are the two most popular utilities for manual analysis of EAP frames. The focus of this portion will be on techniques used to determine valuable pieces of information regarding EAP by using Wireshark. The user must begin by determining whether the packet capture contains any EAP information. This is done by viewing the protocol hierarchy under Statistics > Protocol Hierarchy.

This will show the user the breakdown by protocols of the currently loaded frames. The attacker should look for "802.1X Authentication" under Logical-Link Control in IEEE 802.11 wireless LAN section. If this section is present, then the packet capture contains EAP information; next the user must determine the EAP type in use.



Display filter: none

Protocol	% Packets	Packets	% Bytes	Bytes	Mbit/s
Frame	100.00 %	56	100.00 %	3591	0.113
IEEE 802.11 wireless LAN	100.00 %	56	100.00 %	3591	0.113
IEEE 802.11 wireless LAN management frame	7.14 %	4	13.67 %	491	0.015
Logical-Link Control	42.86 %	24	65.72 %	2360	0.074
802.1X Authentication	42.86 %	24	65.72 %	2360	0.074
Data	8.93 %	5	14.20 %	510	0.016

Figure 1: Wireshark Protocol Hierarchy

After the user has determined that EAP data is present, they use the display filter "eap" to display only the frames within Wireshark containing EAP. The EAP type used by the network can be identified under the 802.1X Authentication Protocol > Extensible Authentication Protocol > Type. If the entire EAP authentication sequence is present, the user must find the starting EAP Identity packet, marked as "Response, Identity [RFC3748]" in the Info field of Wireshark. This particular frame almost always contains the valid identity string of the client. This identity string can be found in the "Packet Details" pane, under 802.1X Authentication Protocol > Extensible Authentication Protocol > Identity. For protocols that use usernames and passwords such as PEAP and TTLS, this can be used to enumerate a list of valid usernames for the target network. On networks using LEAP the "Request" packets can be analyzed to enumerate a list of users for the network.

EAPeak

EAPeak is a tool written in the popular scripting language Python. It takes advantage of the robust packet injection framework provided by Scapy, to parse 802.11 frames from either PCap formatted files or, in a future release, directly from a wireless interface. EAPeak uses the Scapy libraries to parse the data into a Packet object that then can be manipulated using Python to retrieve information regarding EAP authentication that is valuable to an attacker.

When EAPeak is installed using the setup.py script that is included, Python places the eapeak file into a bin directory within the user's path. This is the main file that is executed when a user proceeds to invoke "eapeak" from the command line. When the main EAPeak file is loaded, it begins by importing various python libraries for the core functionality. The two non standard libraries that it requires are the latest versions of Scapy and EAPeak. The EAPeak library itself also is installed to the proper location by Python during the initial installation process.

The EAPeak library is divided into three main files: clients.py, networks.py, and parse.py. Each of these files contains necessary objects for the proper execution of the EAPeak program, and will be described in the following paragraphs.

clients.py

The clients.py file contains the Python class, WirelessClient, which is the object representing each client associated with an EAP enabled wireless network. EAPeak does not store clients that have not been affiliated with an EAP authentication type in these objects, meaning that a user will not see a complete list of associated clients while using EAPeak. This object stores the different attributes such as the client's MAC address and the BSSID that it is associated with, along with a Python list object containing all of the EAP types that EAPeak has determined it supports. The WirelessClient object also contains various methods for the core routines to add and retrieve these attributes. The show() method is responsible for parsing the object's attributes and returning a string suitable for presenting to the end user.

networks.py

Similar to the clients.py file, this contains the WirelessNetwork object. The WirelessNetwork object is created whenever a new network is discovered. It contains the SSID, as well as the BSSIDs that are broadcasting the network. The WirelessNetwork maintains a store of discovered EAP types similar to the WirelessClient object. The WirelessNetwork also is the object responsible for storing the WirelessClient objects for each client that has been associated with it.

parse.py

This is the file that contains the main routines and objects necessary to parse the Scapy Packet objects into usable data. The EAPeakParsingEngine object is responsible for the core functionality of the program. This uses Scapy's "rdpcap" function to iterate over packets from a stored PCap file, and parse them with the parseWirelessPacket() function from libeapeak. This function currently does not rely on any order of packets making it effective even if packet order has not been maintained which is important when parsing multiple PCap files.

The parseWirelessPacket() function determines networks from frames that contains one of the following layers: Beacons, Probe Responses, Association Requests, or any EAP layer. When a frame is found with the later type, the SSID cannot be determined and the network is considered to be "orphaned." It is by using the orphan datastores within the engine object that data that cannot be mapped to a human readable ESSID is not lost. If in a later packet one of the former three types is discovered, the orphaned data is merged into a WirelessNetwork object identified by the human readable SSID.

Finding the EAP in the Haystack

When an EAP layer is found within a frame, the EAPeak parsing engine will proceed to harvest useful information based on the EAP type. Some of the most useful pieces of data that can be extracted from EAP are the various identities used by the clients. When an EAP layer is found with a type of 1 (Identification) and a code of 2 (Response), it contains a string that is used to identify the client to the access point. This identity is often a username that can be used by an attacker for a variety of attacks. EAPeak will pull the identification strings from EAP types that support it being passed in clear text, such as LEAP.

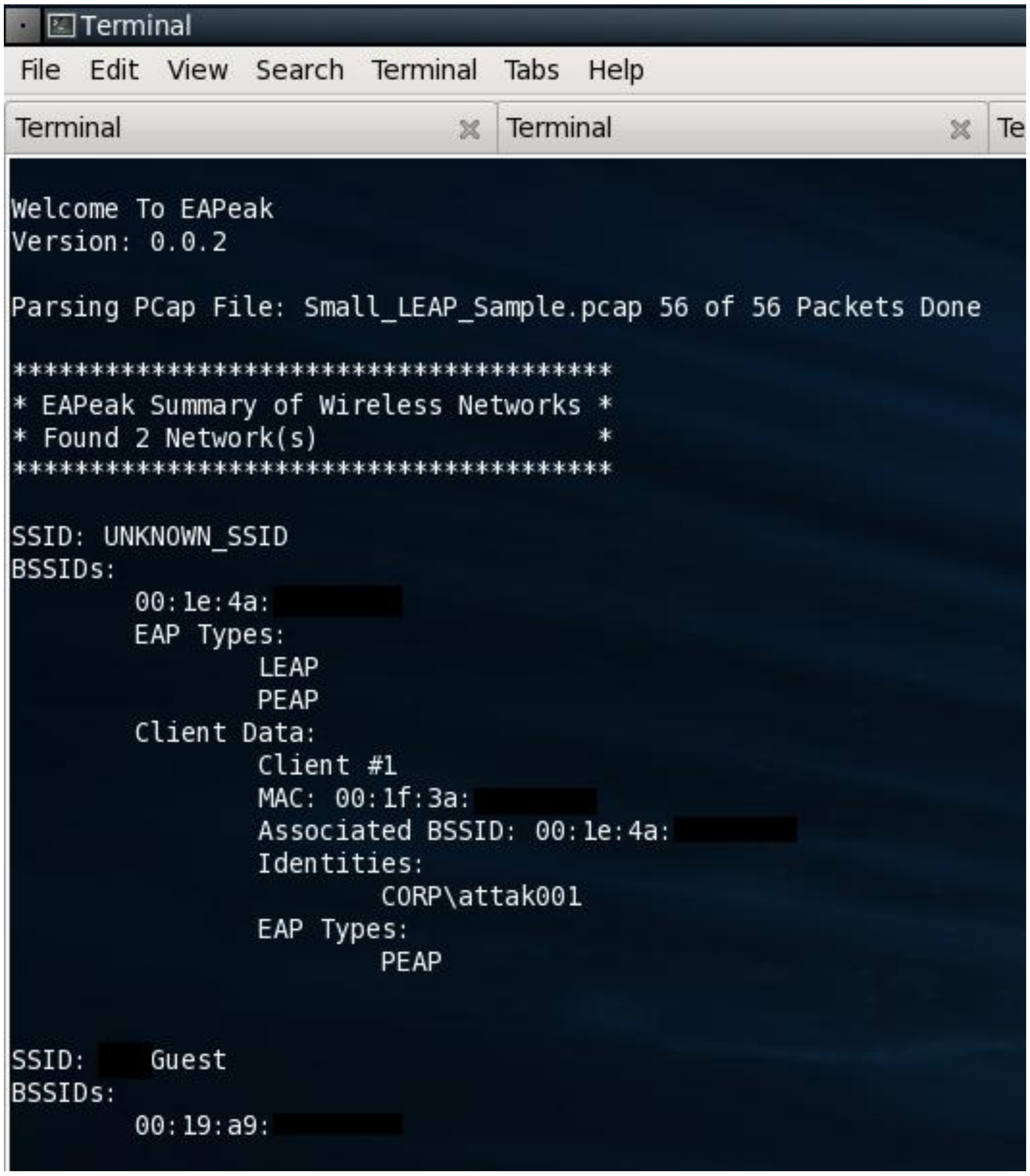
After having received the identity string, the access point will begin trying to authenticate the client using an EAP type of the access point's choosing. At this point the client either responds by continuing to authenticate using this AP-chosen EAP type or replies with a "Legacy Nak" (EAP type #3), effectively rejecting the access point's desired authentication type.

RFC 3748, which dictates the implementation of the EAP protocol, states that the EAP type 3 be used for the client to revoke the access point's chosen EAP type and list one or more authentication types that it supports in an attempt to negotiate a common method with the access point. When EAPeak notices the transaction of a Legacy Nak, all of the types within the response are added to the list of supported types within the client object. This is important to note because an attacker can use this information in determining what authentication types would prove to be valid attack vectors when targeting the particular client in question.

In order for a wireless network to be registered as a network object within EAPeak, a frame containing a Beacon, Probe Response, Association Request, or EAP layer must be found. Furthermore, EAPeak must parse the frames containing the EAP authentication sequence in order to determine an EAP enabled client.

When the EAPeak parsing engine determines that a frame contains an EAP frame, EAPeak uses one of three functions contained within the parse.py module to determine the source, destination, and BSSID. EAPeak determines whether or not a frame is coming from a client or access point by comparing the values returned from getSource() and getDestination() to the returned address of the getBSSID() function. This is important information for EAPeak because knowing whether the data is going to or from the client impacts what pieces of information may be available.

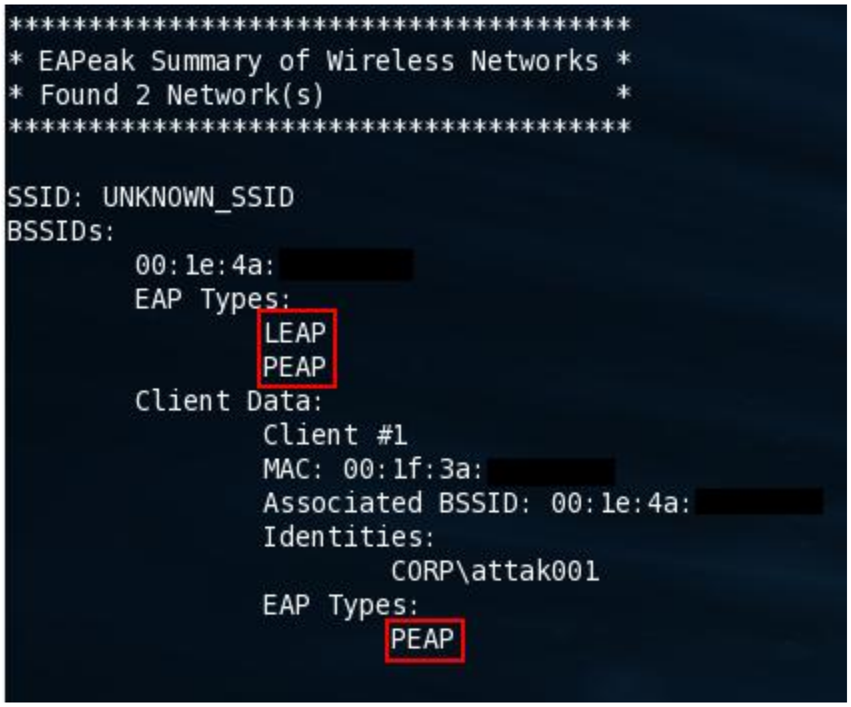
The final result of running EAPeak with a PCap file can be seen in Figure 2, where a network has been detected without a corresponding ESSID.



```
Welcome To EAPeak
Version: 0.0.2

Parsing PCap File: Small_LEAP_Sample.pcap 56 of 56 Packets Done

*************************************
* EAPeak Summary of Wireless Networks *
* Found 2 Network(s)                  *
*************************************

SSID: UNKNOWN_SSID
BSSIDs:
        00:1e:4a:[redacted]
        EAP Types:
                LEAP
                PEAP
        Client Data:
                Client #1
                MAC: 00:1f:3a:[redacted]
                Associated BSSID: 00:1e:4a:[redacted]
                Identities:
                        CORP\attak001
                EAP Types:
                        PEAP


SSID: [redacted] Guest
BSSIDs:
        00:19:a9:[redacted]
```

Figure 2: The output of a parsed PCap file

With this information the attacker can clearly see what clients are connected to this network, what their MAC address is, and which AP which with they are associated. Already the attacker has enough information to use the Air Crack suite to de-authenticate the client to obtain another handshake. The attacker also can see (as marked in red, in Figure 3) that the access point supports using both LEAP and PEAP; however, the client supports only PEAP. This is a real world example of what the attacker would see when the client rejects the AP's original efforts to use LEAP as the authentication method. In this case the client responded with a Legacy Nak, and the authentication proceeded, using PEAP. Had the client included additional methods within their Legacy Nak message, these types would have been added to the EAP types supported by the client, but only the type that the access point agreed on would be added to the AP's list of EAP types. This is because the additional methods proposed by the client cannot be verified to be supported by the access point. If, however, the client supports a method with a valid attack vector, then it is irrelevant whether the access point supports it or not because the attacker can produce their own access point which supports the vulnerable method.



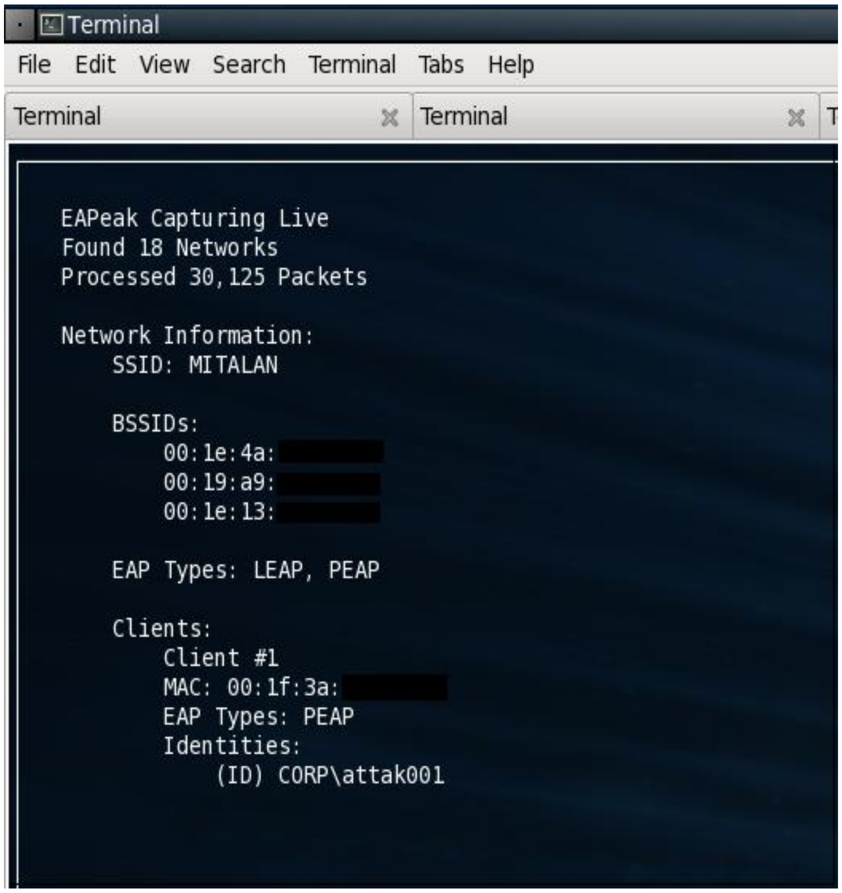
```
*************************************
* EAPeak Summary of Wireless Networks *
* Found 2 Network(s)                  *
*************************************

SSID: UNKNOWN_SSID
BSSIDs:
        00:1e:4a:[redacted]
        EAP Types:
                LEAP
                PEAP
        Client Data:
                Client #1
                MAC: 00:1f:3a:[redacted]
                Associated BSSID: 00:1e:4a:[redacted]
                Identities:
                        CORP\attak001
                EAP Types:
                        PEAP
```

Figure 3: Example of conflicting EAP authentication configuration.

Future Releases

In one of the largest changes planned, EAPeak will rely on modifying the Scapy framework to parse EAP data. Currently Scapy will not parse the Identification strings or other useful data from EAP frames. Scapy also will not parse EAP authentication types such as PEAP, LEAP, and EAP-Fast. In the near future patches will be released for the Scapy framework to implement these layers. By integrating this functionality into the Scapy framework, EAPeak will have access to this information within the native Scapy Packet object. This will shift the parsing from EAPeak to the Scapy framework. Adding this ability to Scapy will allow other programs beyond EAPeak to access this functionality. Additionally adding the EAP authentication layers to the Scapy framework also will allow users to easily inject wireless frames containing EAP authentication information. This functionality will allow future versions of EAPeak to move from an analysis tool to an attack tool.

This user interface also will undergo numerous changes in future releases. The user interface will be changed to utilize the Python Curses library to facilitate real time reporting of important information when EAPeak is used in live mode. An example of the Curses interface is shown in figure 4.



```
EAPeak Capturing Live
Found 18 Networks
Processed 30,125 Packets

Network Information:
    SSID: MITALAN

    BSSIDs:
        00:1e:4a:[redacted]
        00:19:a9:[redacted]
        00:1e:13:[redacted]

    EAP Types: LEAP, PEAP

    Clients:
        Client #1
        MAC: 00:1f:3a:[redacted]
        EAP Types: PEAP
        Identities:
            (ID) CORP\attak001
```

Figure 4: The Beta Curses interface

About the Authors

Matt Neely (CISSP, CTGA, GCIH, and GCWN) is the Profiling Team Manager at SecureState, where he leads a Team which performs traditional penetration tests, physical penetration tests, web application security reviews, and wireless security assessments. His research interests include the convergence of physical and logical security, lock and lock picking, cryptography, and all things wireless. Matt is a host of the Security Justice podcast.

Spencer McIntyre is a security consultant at SecureState where he focuses on penetration testing and tool development. He spends most of his free time focusing on his primary interests of vulnerability research and exploit development.

About SecureState

SecureState is a management consulting firm specializing in information security. SecureState draws upon the skill sets of its team members, and their extensive experience in government, private-sector, and Big X consulting firms. SecureState is comprised of five specialties: Advisory Services, Audit and Compliance, Profiling, Risk Management, and Business Preservation Services. Whether you need to be compliant with industry regulations, need your network tested for security, desire to build a solid security program, or have experienced a data breach, SecureState has the expert team that you need.