NAC@ACK

Hacking the Cisco NAC Framework

Michael Thumann, mthumann@ernw.de
&
Dror-John Roecher, droecher@ernw.de

Version 1.0 – March 9th 2007

Marketing Buzz – What is Cisco NAC?

Looking back at the evolution of security one can easily make out technologies that become en vogue for some time and then transform from “hot-stuff” into standard-security technologies. Just think of firewall, S2S-VPNs, RAS-VPNs, IDS, IPS or SSL-VPN solutions, to name a few. Moving away from perimeter-defense and taking security into the depth of networks, security has been concentrating on endpoints for quite some time now. Virusscanner, Anti-Spyware/Anti-Phishing and Desktop-Firewalls are some popular examples of Endpoint-Security-Technologies nowadays commonly applied to Enterprise-Desktops.

Besides Endpoint Security there has been a big fuzz on “compliance solutions”, though everyone seems to understand compliance in a slightly different way. Compliance to BCP, or to external regulations (just think about SOX or Basel-II) or even compliance with internal policies (is the antivirus up-to-date?) – as you can see there are many different flavours of compliance.

Now try to imagine what a vendor could achieve by combining “Endpoint Security” and “Compliance” into a single concept or solution – as a vendor you would have two cows to milk: The “IT” department and the “Compliance/Audit” department. Additionally you could try to make this a really big source of additional revenue, because a solution like that could be as pervasive as you wish – creating the need to upgrade/replace all network-equipment, installing new management-servers, selling client-software et cetera. Of course you would need to be a big vendor to be able to push your proprietary solution upon the market, like e.g. Cisco Systems. As you may have noticed, Cisco has been quite heavy on the marketing of the “Self Defending Network” [we will refrain from commenting on that term] and a core component of that is “Network Admission Control”, or simply NAC.

Cisco’s NAC solution has the goal to improve security by allowing only compliant clients admission to the network – the compliance is defined by a set of self-defined policies (e.g. antivirus-signatures are current and antivirus is active). Clients failing to comply with the policies can be granted restricted access or denied access altogether. Sounds pretty cool – it actually is and it works quite well, but the design is flawed which enables an attacker to access a NAC-protected networks without being compliant under certain circumstances. Under what circumstances and how to get that access is what will be detailed in this paper. But first a thorough understanding of the technical operations and components involved is needed – so let’s stop the marketing buzz now and dig into the technology.

A look under the hood – NAC technical primer

The Cisco NAC solution encompasses four distinct components playing together to check a client for compliance and to enforce access-restrictions on the client based on the results of the checks. Naturally a new solution like NAC will come along with a new acronym-soup and terminology. Figure 1 depicts the relationship between the components and the protocols used for communication.

1. Client-side software: For NAC to check the state of a client, a piece of software is required to run on the client and this software, in its most basic form, is the “Cisco Trust Agent” (CTA). The CTA communicates via EAP and supplies so called “credentials” – basically attribute-values-pairs conveying information about the state of the client to a backend policy-server. The CTA itself can provide some basic credentials (OS-Type & Version, etc – see Table 3 for a complete list) but if further credentials are supposed to be checked, additional NAC-aware software is needed on the client to plug-in to the CTA and provide the CTA with the needed information.

2. The “Enforcing Device”/Network Access Device (NAD): The Network Access Device is the enforcing point on the network and it has two tasks. It relays the EAP-messages from the CTA as RADIUS-messages to the Cisco Secure ACS and back to the client and it enforces the policy determined by the Cisco Secure ACS. Routers, Switches, Firewalls, VPN-Concentrators and WLAN-Access-Points can act as NAC-enabled devices as long as they run up-to-date software and are “made by Cisco”.

3. Cisco Secure ACS: The core component of Cisco’s NAC is the “Cisco Secure Access Control Server” (ACS). The ACS is a RADIUS server which serves multiple functions.

    - It accepts client-credentials via the NAD.
    - It checks the credentials against a local policy
    - It may check 3rd-party policy servers, if additional NAC-enabled applications are running on the client.
    - From the credentials provided and policies configured, access-restrictions are derived and communicated to the NAD and to the client.

4. Optional Backend-Servers: If additional security-software, e.g. antivirus, is to be included in NAC, then not only need it be installed on the client, but a policy-server for that application may be needed, too. The ACS needs to be made aware of this external policy server and it queries it using the “Host Credential Authorization Protocol” (HCAP) – basically “XML over HTTPS”. This third-party policy server is not looked at within this paper or the accompanying talk.

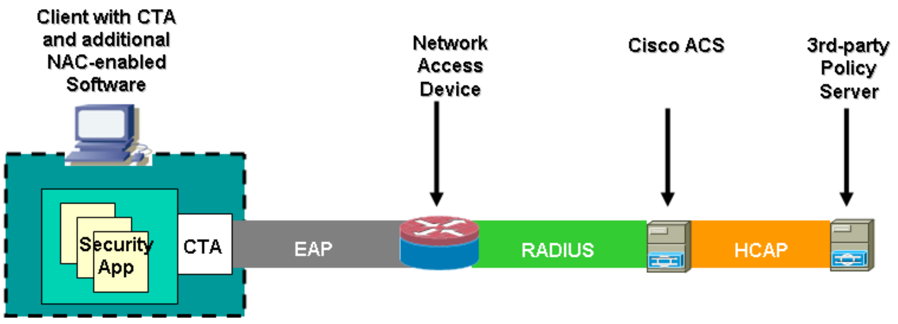


Figure 1 Components of a Cisco NAC Network

Terminology

Before delving into more technical details the used terminology shall be introduced. The information provided by the client to the ACS is called “posture credentials”. For each NAC-enabled application the ACS derives an “application posture token” (APT). This posture token is one out of “healthy”, “checkup”, “transition”, “quarantine”, “infected” or “unknown”. These tokens have a well-defined meaning:

- “Healthy”: fully compliant with the admission policy for the specified application.
- “Checkup”: partial but sufficient compliance with the admission policy, no need to restrict access, a warning to the user may be issued.
- “Transition”: either during boot-time, when not all necessary services have been started or during an audit-process for clientless hosts[1], temporary access-restrictions may be applied.
- “Quarantine”: insufficient compliance with the admission policy, network access is usually restricted to a quarantine/remediation segment.
- “Infected”: active infection detected, usually most restrictive network access even up to complete isolation.
- “Unknown”: a token can not be determined or no CTA installed on client. This may lead to partial access (guest-vlan & internet-access for example).

From all APTs a system posture token (SPT) is deduced, this SPT corresponds to the APT which will grant the least access to the client. Based on the SPT the access-restrictions are chosen and sent back to the NAD to be enforced. All APTs and the SPTs are also communicated back to the client, where software may use this information to inform or warn the user or to change an applications’ behavior.

There are 3 different “flavours” of NAC relating to different access-vectors of the client bringing along different methods for restricting access to the network.

1. Layer3-IP: With NAC Layer3-IP the access-restrictions are implemented as IP-ACLs on a Layer-3 device (e.g. a Router or a VPN-Concentrator/Firewall). The communication takes place using PEAP over EAP over UDP (EoU).
2. Layer2-IP: Layer2-IP NAC enforces access-restrictions as IP-ACLs on a VLAN-interface of a switch. The communication takes place using PEAP over EAP over UDP (EoU).
3. Layer2-802.1x: Layer2-802.1x uses 802.1x port control to restrict network access – obviously the device enforcing these restrictions is a switch. In this case the EAP-FAST is used in conjunction with 802.1x. This is the only NAC flavour where the client is:

   a. authenticated before being allowed on the network

   b. restricted from communicating with its local subnet

   This is the most secure (and the most elaborate) NAC setup possible, because a working 802.1x infrastructure is assumed to be in place, but because of its “Layer2-nature” it is not feasible on all setups.

Table 1 shows some of the features of the three different variants of NAC and serves as a quick comparison.

[1] Clientless hosts are hosts which have no CTA installed or where the CTA is not operating properly. These fall into three categories: (1) Legitimate systems where the CTA can not be installed, e.g. network-printers, (2) external hosts, e.g. guests or consultants and (3) legitimate systems where the CTA is not operating properly. These systems can be either statically whitelisted (this is done with systems from the first category) or can be “audited” by a vulnerability-scanner, or they may be treated with a “default”-restriction like guest-access.

Feature	NAC-L2-802.1x	NAC-L2-IP	NAC-L3-IP
Trigger	Data Link / Switchport	DHCP / ARP	Routed Packet
Machine ID	Yes	No	No
User ID	Yes	No	No
Posture	Yes	Yes	Yes
VLAN Assignment	Yes	No	No
URL Redirection	No	Yes	Yes
Downloadable ACLs	Cat65k only	Yes	Yes
Posture Status Queries	No	Yes	Yes
802.1x Posture Change	Yes	No	No
Transport EoU/PEAP	No	Yes	Yes
Transport 802.1x / EAP-FAST	Yes	No	No

Table 1Feature Comparison

Each of these different flavours is suitable for different situations and they definitely differ in the security level they provide and how the secure the mechanisms themselves are (as e.g. is obvious from the lack of authentication in two flavours).

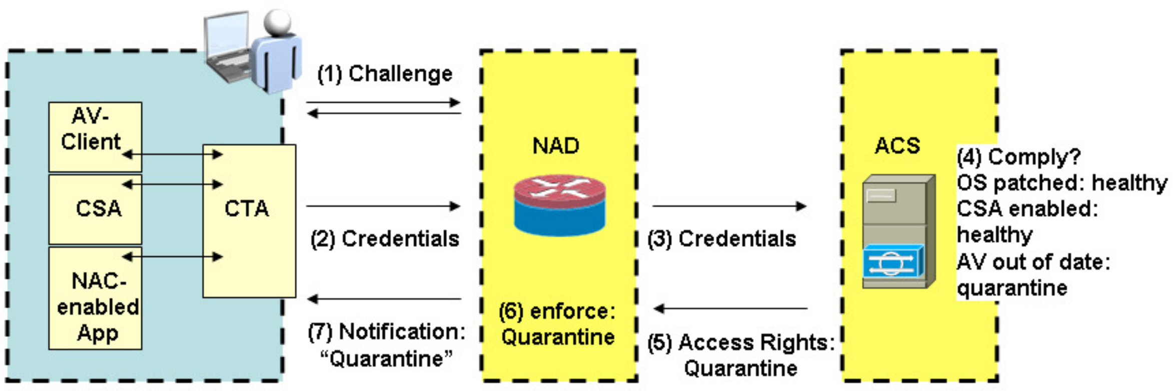


Figure 2 Sample communication-flows in Cisco NAC

More technical Details

Now that the technology has been roughly sketched it is time for some technical details which will become interesting or important later on.

Host Credential Information

The credentials collected on the host and communicated to the ACS can be roughly divided into two different classes. (1) Information gathered through NAC-enabled applications and (2) arbitrary information gathered by the CTA through a scripting interface.

The credentials are transmitted as attribute-value-pairs and are of different value-types:

- Octet-Array, Integer32, Unsigned32, String (UTF-8), IPv4 Address, IPv6 Address, Time (4 Octets), Version (4-x-2 octet-sets)

The attribute-part of the AV-pair is defined by a set of parameters:

- Numerical Vendor ID / Vendor ID Name. The IANA SMI private enterprise ID assignments are used.[2] For example Cisco has ID 9, McAfee 1230, Trendmicro 6101.
- Numerical identifier of the application type (16 Bit value). Some values are reserved:

Application-Type	Application-Type Name	Usage
1	PA	Posture Agent
2	Host / OS	Host information
3	AV	Anti Virus
4	FW	Firewall
5	HIPS	Host IPS
6	Audit	Audit
32768 – 65536		Reserved for "local use" (custom plug-ins or scripts)

Table 2 Application Identifiers

- Numerical Attribute-ID, Attribute Name & Attribute-Type

The following table lists the credentials available through the CTA (version 2.0) without installation of any other NAC-enabled application. This is meant to give an impression on the type of information available for policy-enforcement.

Application-Type	Attribute Number	Attribute Name	Value-Type
Posture Agent	3	Agent-Name (PA-Name)	String
	4	Agent-Version	Version
	5	OS-Type	String
	6	OS-Version	Version
	7	User-Notification	String
	8	OS-Kernel	String
	9	OS-Kernel-Version	Version

[2] For SMI private enterprise assignments check: http://www.iana.org/assignments/enterprise-numbers

Host	11	Machine-Posture-State	1 – Booting, 2 – Running, 3 – Logged in.
	6	Service Packs	String
	7	Hot Fixes	String
	8	Host-FQDN	String

Table 3 CTA included credentials

So a posture-credential can be identified in shorthand as "Vendor : Application-Type : Attribute", e.g. "9:1:8" would translate to Vendor: Cisco (9), Application: Posture Agent (1), Attribute: OS-Kernel and with reference to Table 3 it would expect it to be a string (e.g. "Linux-2.6.4-8-i386-custom").

As mentioned the CTA has two distinct ways of enabling credentials to be collected and transmitted. The first one is "regular posture plug-ins" and the second one is via a "scripting interface". Let's have a short look on how they differ and what they do.

Plug-ins (on Windows) are realized through dll-files (an so-files on Linux) which are located in "%CommonProgramFiles%\PostureAgent\Plugins" (again on Windows) and are configured through corresponding ini-files where the available credentials for the plug-in are described. When installing CTA 2.0 three plug-ins are installed by default:

- Host Posture Plug-in
- CTA Plug-in
- Scripting Plug-in

Third-party applications will install additional plug-ins.

The scripting interface (realized through ctasi.exe and ctascriptPP.dll on the client) can be used to add third-party-applications and self-written assessments to NAC which are not realized as posture plug-ins. Again an ini-file is used to describe the available credentials and an external program is referenced which collects the credentials - the format of the inf-file is the same as for plug-ins. The program is assumed to write the collected credentials into a plaintext "posture-data" file which is read by the CTA (actually the program is expected to call ctasi.exe) and parsed using the information from the inf-file. The inf-files for plug-ins and scripts take the same format.

A sample ini-file (tmabpp.inf) from Trendmicros' OfficeScan will serve to illustrate this process:

```
[main]

dll=tmabpp.dll
PluginName=tmabpp.dll
VendorID=6101
VendorIDName=TrendMicro, Inc
AppList=av

[av]

AppType=3
AppTypeName=Antivirus
AttributeList=attr1,attr2,attr3,attr4,attr5,attr6,attr7,attr8,attr9,attr10,attr11,attr12,attr13,attr14
attr1= 1, Unsigned32, Application-Posture-Token
attr2=2, Unsigned32, System-Posture-Token
attr3=3, String, Software-Name
attr4=4, Unsigned32, Software-ID
attr5=5, Version, Software-Version
attr6=6, Version, Scan-Engine-Version
attr7=7, Version, Dat-Version
attr8=8, Time, Dat-Date
attr9=9, Unsigned32, Protection-Enabled
attr10=10, String, Action


attr11=32768, String, OSCE-Srv-Hostname
attr12=32769, OctetArray, Client-GUID
attr13=32770, Ipv4Address, Client-IP
attr14=32771, OctetArray,  Client-MACddd
```

The name of the plug-in. In case of a script this would be ctascriptPP.dll and the vendor-id would be "Cisco" for scripts.

Official Credentials

Private Credentials from the Vendor

And a sample posture-data file which is created as output of a script and passed to the CTA:

```
[attr#0]
vendor-id=9
vendor-name=Cisco
application-id=61440
application-name=ERNW-NAC-Script-1
attribute-id=32768
attribute-name=Script-Name
attribute-profile=in
attribute-type=string
attribute-value=Script "posture_file_01"
[attr#1]
vendor-id=9
vendor-name=Cisco
application-id=61440
application-name=ERNW-NAC-Script-1
attribute-id=32769
attribute-name=Script-Run-Counter
attribute-profile=in
attribute-type=unsigned integer
attribute-value=17
[attr#2]
vendor-id=9
vendor-name=Cisco
application-id=61440
application-name=ERNW-NAC-Script-1
attribute-id=32770
attribute-name=Host-IP-Address
attribute-profile=in
attribute-type=ipaddr
attribute-value=196.168.100.200
[attr#3]
vendor-id=9
vendor-name=Cisco
application-id=61440
application-name=ERNW-NAC-Script-1
attribute-id=32772
attribute-name=Script-Date
attribute-profile=in
attribute-type=date
attribute-value=1077771601
[attr#4]
vendor-id=9
vendor-name=Cisco
application-id=61440
application-name=ERNW-NAC-Script-1
attribute-id=32773
attribute-name=Script-Version
attribute-profile=in
attribute-type=version
attribute-value=0.1.0.0
```

A list containing detailed descriptions of all attributes and their actual values. These must match the corresponding inf-file of the script.

Vendor is always Cisco for scripts and Application-ID is from "local use" range.

This data-file is created by the script which then invokes ctasi.exe.

Ctasi.exe parses the script and relays the credentials via the ctascriptPP.dll plug-in to the regular CTA communication process.

The architecture of the CTA (as shown in Figure 3) clarifies the relationship between the different components. The communication layer accepts posture-credentials (EAP-TLV) from the broker and passes this information to lower variant-specific transport-oriented modules (EoU/802.1x). The posture-plugin API and scripting interface supply information from plug-ins or from scripts to the broker/communication-layer.

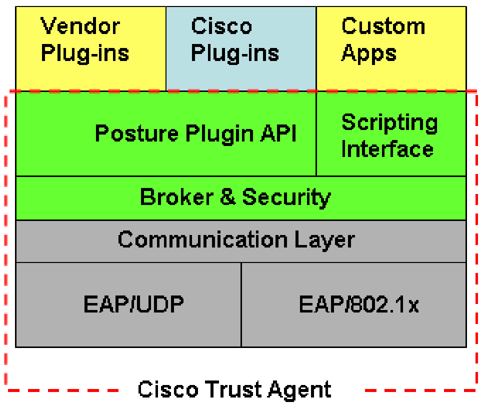


Figure 3 Architectural Overview of the CTA

A simple sample Lab-setup for NAC-L3-IP

To get a feeling for the operations of NAC a brief description of a NAC-L3 setup will be described. This is the description of the LAB actually used for the research presented in this paper and for developing the attacks shown in our presentation. Figure 4 depicts a generalized view of the environment, the ACS is running in the “NAC protected Core”. Users can access the Office-Environment through RAS-VPN but need to pass “NAC protection” in order to access the “holy core”. The choice of “NAC-flavour” for this scenario is easy – obviously Layer2-802.1x will not work as the clients are on the internet and are not connected to a local switch in the office. The same holds true for L2-IP and additionally the NAD never sees ARP-requests or DHCP-requests from the client. So the only possible flavour for this setup is NAC-L3-IP.

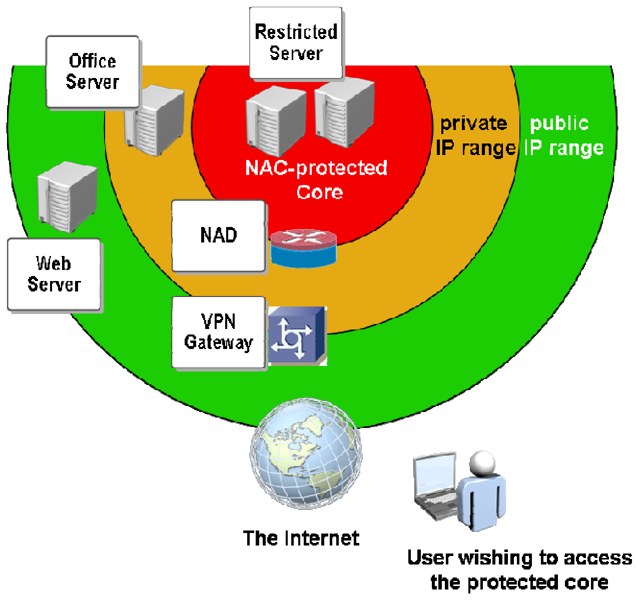
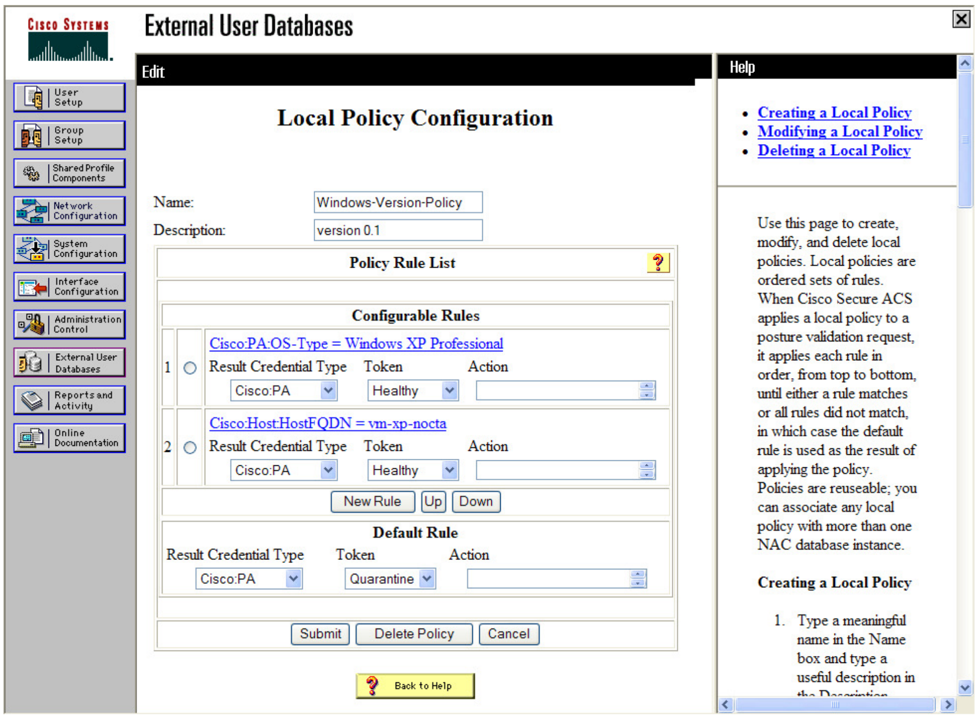


Figure 4 Sample NAC-L3 Setup

The setup was tested with a multitude of different policies in order to understand the packet-format and the behaviour of the application. These policies were kept as simple as possible. Please see Figure 5 for an example policy used in the test-scenarios..



Local Policy Configuration

Name: Windows-Version-Policy
Description: version 0.1

Policy Rule List

Configurable Rules

1	Cisco:PA:OS-Type = Windows XP Professional	
	Result Credential Type: Cisco:PA	Token: Healthy	Action:
2	Cisco:Host:HostFQDN = vm-xp-nocta	
	Result Credential Type: Cisco:PA	Token: Healthy	Action:

Default Rule

Result Credential Type	Token	Action
Cisco:PA	Quarantine	

- Creating a Local Policy
- Modifying a Local Policy
- Deleting a Local Policy

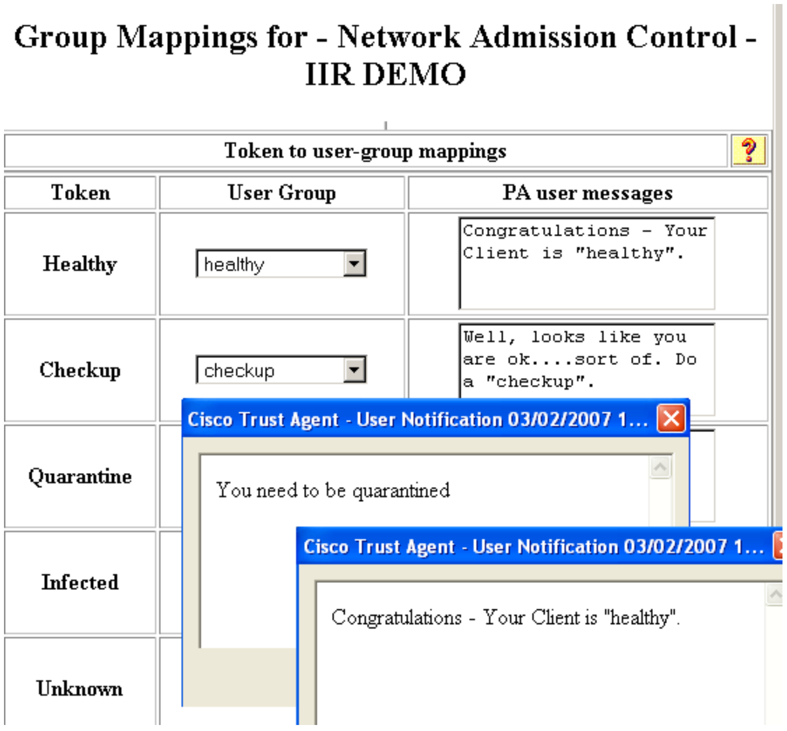
Use this page to create, modify, and delete local policies. Local policies are ordered sets of rules. When Cisco Secure ACS applies a local policy to a posture validation request, it applies each rule in order, from top to bottom, until either a rule matches or all rules did not match, in which case the default rule is used as the result of applying the policy. Policies are reuseable; you can associate any local policy with more than one NAC database instance.

Creating a Local Policy

1. Type a meaningful name in the Name box and type a useful description in

Figure 5 A sample policy which will result in an APT for PA of „healthy“ if the client is running Windows XP and has a hostname of „vm-xp-nocta“. The APT will be ”quarantine” if at least one of the tests fail.

The client can be notified of the results of the assessments. The notification is dependent on the SPT and is configured on the ACS (see Figure 6).


Group Mappings for - Network Admission Control - IIR DEMO

Token to user-group mappings

Token	User Group	PA user messages
Healthy	healthy	Congratulations - Your Client is "healthy".
Checkup	checkup	Well, looks like you are ok....sort of. Do a "checkup".
Quarantine		
Infected		
Unknown		

Cisco Trust Agent - User Notification 03/02/2007 1...

You need to be quarantined

Cisco Trust Agent - User Notification 03/02/2007 1...

Congratulations - Your Client is "healthy".

Figure 6 User notification setup in ACS and CTA client notification popup.

The configuration of the NAD (in our case a Cisco 3640) is quite simple. An "admission policy" is created and bound to the external interface. On the same interface an ACL is in place which allows

```
version 12.4
hostname nac-lab-nad
aaa new-model
aaa authentication eou default group radius
ip domain name ernw.de
ip admission name NAC-LAB eapoudp
!
!
!
eou clientless username clientless
eou clientless password Cisco
eou allow clientless
!
interface FastEthernet3/0
  description "protected core"
  ip address 192.168.81.33 255.255.255.224
!
interface FastEthernet3/1
  description "office network"
  ip access-group 100 in
  ip address 192.168.81.66 255.255.255.224
  ip admission NAC-LAB
!
ip http server
!
access-list 100 remark ****** NAC-ACL **********
access-list 100 permit ip any host 192.168.81.34
access-list 100 permit udp any any
!
radius-server host 192.168.81.34 auth-port 1645 acct-port 1646 key 7 SECRETKEYREMOVED
radius-server vsa send authentication
!
end
```

Authentication for EOU via default-radius group

Defining an admission policy for usage with EOU

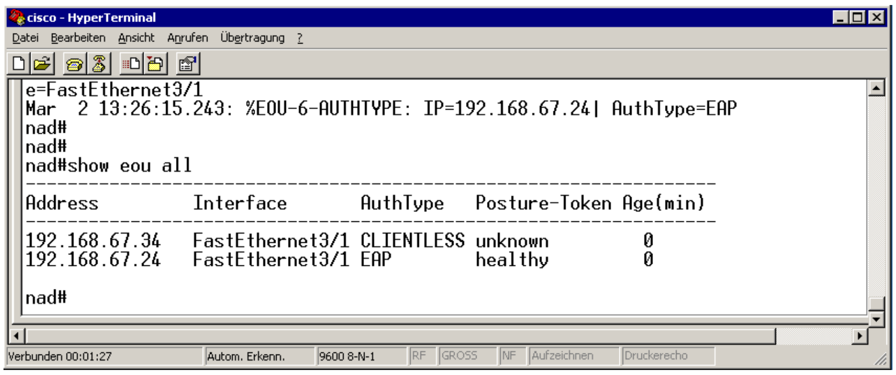
Configuration for systems w/o CTA

ACL which will need to allow at a minimum EOU traffic to always pass.

Binding the admission policy to the "external" interface.

at least EoU traffic (UDP port 21862 to the ACS) and if access-restrictions for a client are configured as ACLs on the ACS, the ACL on the interface will be dynamically expanded with these downloadable ACLs. Additionally an optional configuration for clientless hosts can be configured. In our case clientless hosts are allowed and are authorized with username "clientless" and password "cisco" – this user (clientless) is configured on the ACS with a downloadable ACL for restricting access of these clients on the network.

When traffic passes the NAD from the office-network towards the core, the traffic is intercepted by the admission policy and the client is asked to provide the credentials. The NAD keeps track of all NAC-assessments and the current SPT of all clients. As can be seen in Figure 7 an agentless host (192.168.67.34) is treated according to the above configuration (authenticated as "clientless" with a password against the ACS user-database and hosts with CTA installed are validated via EAP. In the depicted case the CTA-host (192.168.67.24) has successfully passed all policy-checks and is assigned a SPT of "healthy".


cisco - HyperTerminal
Datei Bearbeiten Ansicht Anrufen Übertragung ?
Verbunden 00:01:27 Autom. Erkenn. 9600 8-N-1 RF GROSS NF Aufzeichnen Druckerecho

```
e=FastEthernet3/1
Mar  2 13:26:15.243: %EOU-6-AUTHTYPE: IP=192.168.67.24| AuthType=EAP
nad#
nad#
nad#show eou all
-----------------------------------------------------------------
Address         Interface        AuthType   Posture-Token Age(min)
-----------------------------------------------------------------
192.168.67.34   FastEthernet3/1 CLIENTLESS unknown         0
192.168.67.24   FastEthernet3/1 EAP        healthy         0

nad#
```

Figure 7 Output of „show eou all" on the NAD

Communication Flow for NAC-Layer3-IP

NAC-Layer3-IP uses PEAP over EAP over UDP as a transport-mechanism for the credentials. PEAP establishes a TLS-tunnel between the client and the ACS. Within this tunnel client can be optionally authenticated - which is not the case in Cisco NAC. Figure 8 shows a generic communication flow for NAC-Layer3-IP and Figure 11 shows the communication flow including the information that is carried within each step.

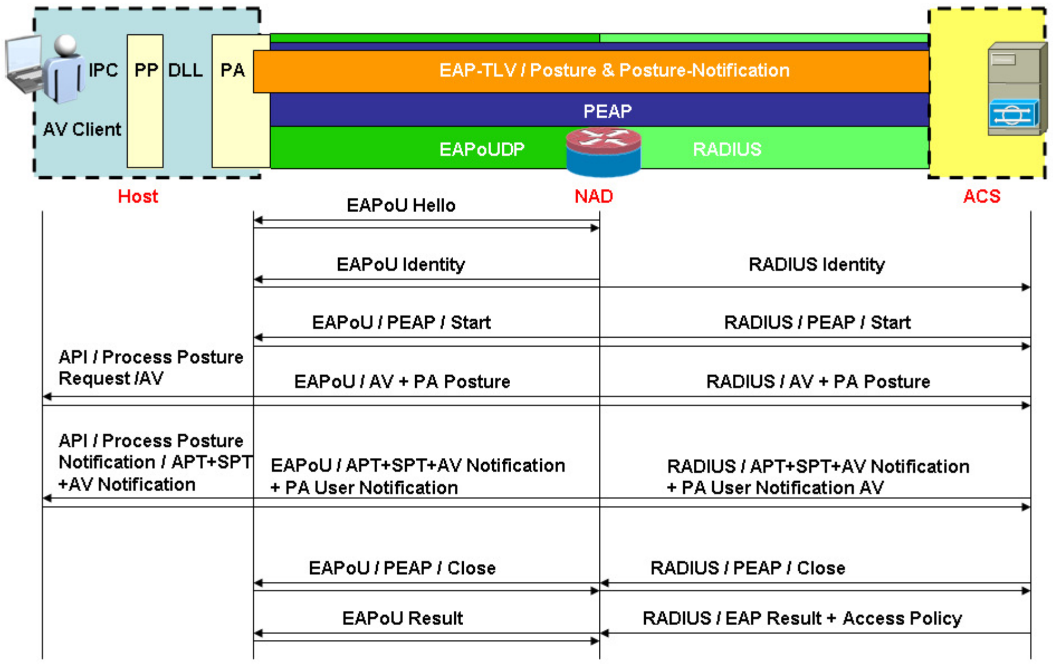

IPC
PP
DLL
PA
AV Client
EAP-TLV / Posture & Posture-Notification
PEAP
EAPoUDP
RADIUS
Host
NAD
ACS
EAPoU Hello
EAPoU Identity
RADIUS Identity
EAPoU / PEAP / Start
RADIUS / PEAP / Start
API / Process Posture Request /AV
EAPoU / AV + PA Posture
RADIUS / AV + PA Posture
API / Process Posture Notification / APT+SPT +AV Notification
EAPoU / APT+SPT+AV Notification + PA User Notification
RADIUS / APT+SPT+AV Notification + PA User Notification AV
EAPoU / PEAP / Close
RADIUS / PEAP / Close
EAPoU Result
RADIUS / EAP Result + Access Policy

Figure 8 Detailed Communication Flow in NAC Layer-3 IP[3]

The Cisco NAC for Layer3-IP and Layer2-IP is built on PEAPv1 and EAP-TLV. The PEAP communication uses two distinguished phases. Phase 1 establishes a secure tunnel using EAP-TLS and authenticates the server using a server-certificate. The second phase includes an optional client-authentication (not implemented in NAC) and the exchange of arbitrary information – in the case of NAC this arbitrary information is composed of posture data and posture notifications using EAP-TLV as a means of encoding that information. Figure 10 shows the frame format for Cisco PEAP

[3] Citing from the "Implementing Network Admission Control Phase One Configuration and Deployment" guide (available on http://www.cisco.com): „ Note that the router acts as a pass-through device at this point; it does not proxy any part of the PEAP session but merely reencapsulates the PEAP packets from UDP to RADIUS."

(PEAPv1) and Figure 12 & Figure 13 show the format for EAP-TLV and EAP-TLV Vendor specific frames respectively.

In Cisco NAC PEAP is carried as the payload of EAP over UDP (EAPoU), leading to a generic frame-format shown in Figure 9.

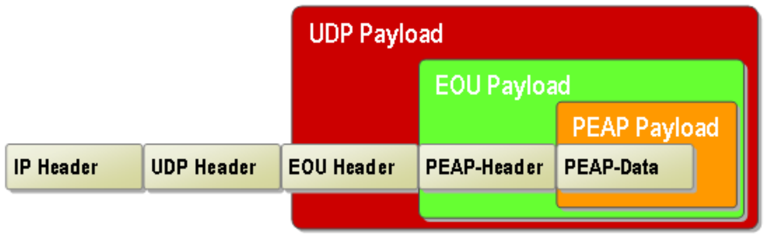


Figure 9 Generic Frame Format

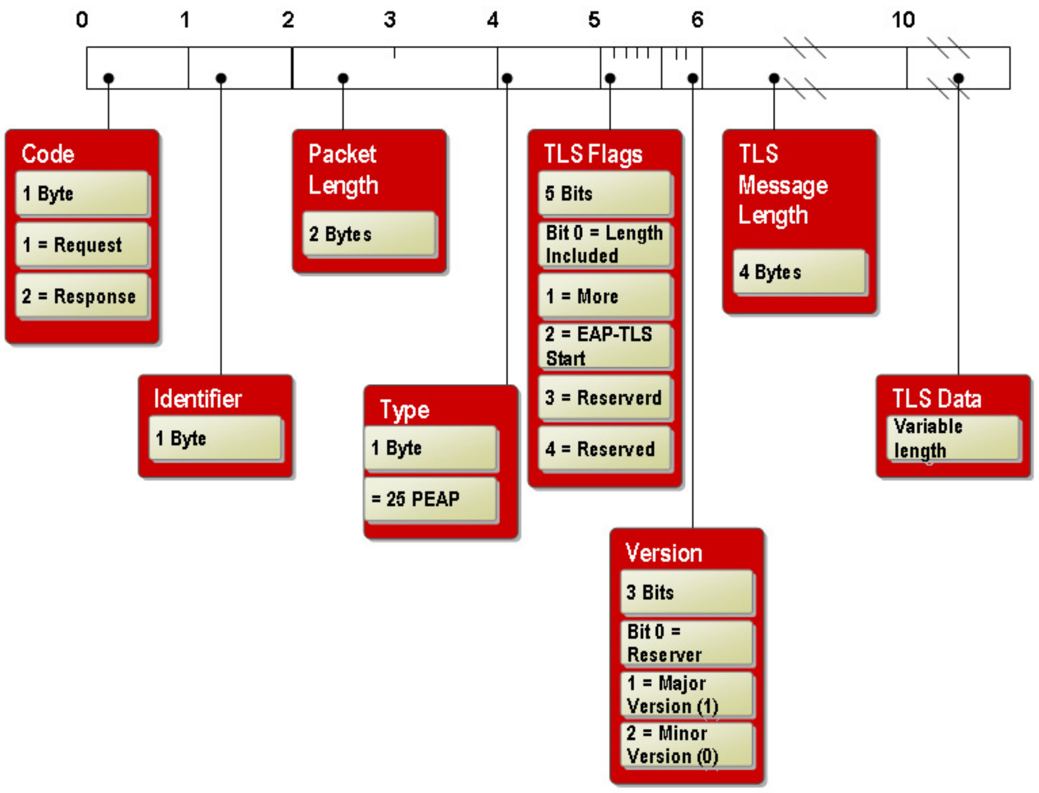
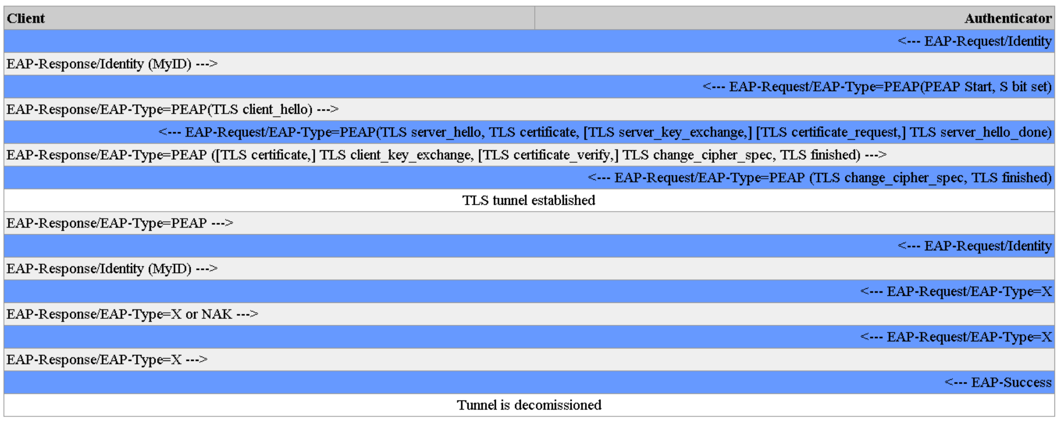


Figure 10 Cisco PEAP Frame Format

PEAP uses a multiple packets to establish the tunnel and ensure secure communications. A rough overview was given in Figure 8 – a more detailed communication flow for PEAPv1 tunnel establishment and decommissioning is given in Figure 11.



Client	Authenticator
	<--- EAP-Request/Identity
EAP-Response/Identity (MyID) --->	
	<--- EAP-Request/EAP-Type=PEAP(PEAP Start, S bit set)
EAP-Response/EAP-Type=PEAP(TLS client_hello) --->	
	<--- EAP-Request/EAP-Type=PEAP(TLS server_hello, TLS certificate, [TLS server_key_exchange,] [TLS certificate_request,] TLS server_hello_done)
EAP-Response/EAP-Type=PEAP ([TLS certificate,] TLS client_key_exchange, [TLS certificate_verify,] TLS change_cipher_spec, TLS finished) --->	
	<--- EAP-Request/EAP-Type=PEAP (TLS change_cipher_spec, TLS finished)
TLS tunnel established	
EAP-Response/EAP-Type=PEAP --->	
	<--- EAP-Request/Identity
EAP-Response/Identity (MyID) --->	
	<--- EAP-Request/EAP-Type=X
EAP-Response/EAP-Type=X or NAK --->	
	<--- EAP-Request/EAP-Type=X
EAP-Response/EAP-Type=X --->	
	<--- EAP-Success
Tunnel is decomissioned	

Figure 11 Detailed view of PEAPv1 packet flow

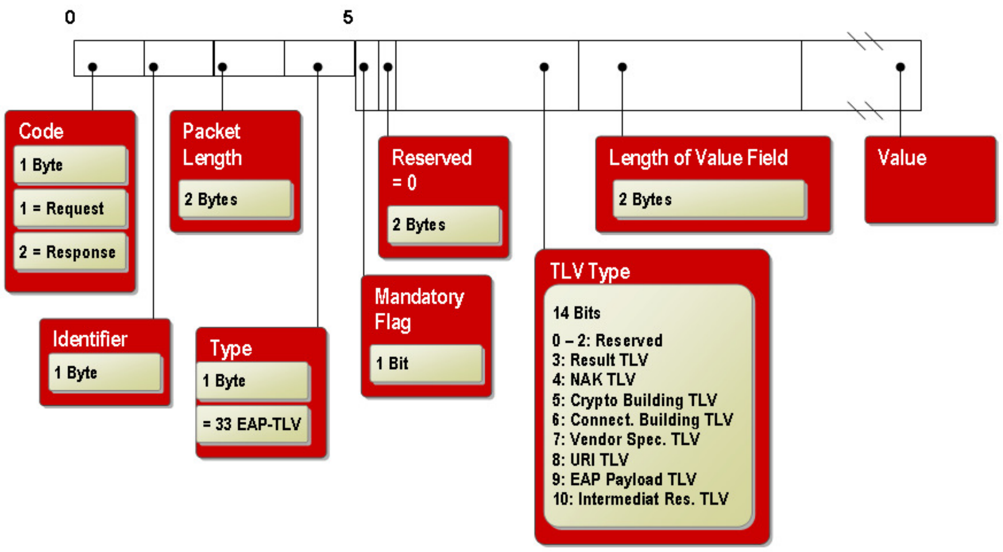


Figure 12 EAP-TLV Frame Format

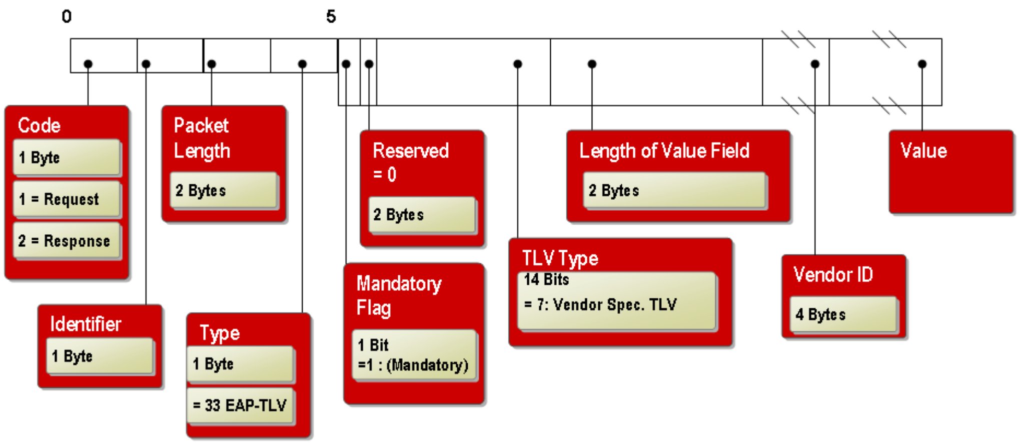


Figure 13 EAP-TLV Vendor Packet Format

Flawed by Design – Analysis of Cisco NAC

Looking at the three different variants of Cisco NAC it is seen, that they differ significantly in the security they provide for the network on at least two details:

	NAC-Layer 3 IP	**NAC Layer 2 IP**	**NAC Layer 2 802.1x**
Client Authentication	No intrinsic Client Authentication. In VPN scenarios there is a "VPN Authentication" which might be considered a "mitigating control".	No intrinsic Client Authentication – and no means of "adding" such on top.	Client Authentication based on 802.1x
Restriction of access on local subnet.	It is not possible to restrict access to the local subnet via NAC.	It is not possible to restrict access to the local subnet via NAC.	Access to local subnet can be denied through "port shutdown" via NAC.

Table 4 Authentication & restrictions for Cisco NAC flavours.

A problem common to all NAC solutions is that the state of the Client is used to derive the access-level (one could also name this "trust level") on the network. Why is that a problem? Because this information (state of client) has to be provided by the client – whom one doesn't trust first place, or else one wouldn't be thinking about using NAC to distinguish between "trustworthy" and "not trustworthy" clients. So the untrusted client is asked to provide credentials in order to judge its trustworthiness. The problem at hand reminds us of the Epimenides paradox – a logical problem named after the Cretan philosopher Epimenides of Kronos. It can be stated like that:

Epimenides was a Cretan who made one statement: "All Cretans are liars."

The next problem (which is a Cisco NAC design specific) is that in two out of three possible variants, there is no intrinsic client authentication whatsoever. We are talking about authorization without authentication. If you wear a police-shirt, NAC treats you as a police-officer without ever asking you for any proof – you look like a police officer, so you have to be one.

The design of Cisco NAC is either flawed or NAC was not developed as a security-mechanism but as something else. We can't help but cite from the Cisco NAC website (http://www.cisco.com/go/nac). There is a section "NAC business benefits" and the first listed benefit is "dramatically improves security". In my opinion it should read "NAC might dramatically improve your security". So we postulate that the design is flawed. Feel free to prove us wrong.

The next obvious question is "is it possible to exploit this design-flaw?" and if the answer to that question is "yes" [which, it is] – than "under which circumstances and how can we exploit it and what could be achieved by that?" The "restrict access to local subnet" we quickly put away as "not interesting" – there is nothing to hack in that respect, it is simply a matter of "plug in your client" and be connected. The fact that the access-level is derived from client-provided information is the far more interesting aspect in NAC. By the way this is not specific to the Cisco solution but is common to most NAC approaches (Consentry being a notable exception to this). And the "inadequate authentication" or as we like to call it "authorization without authentication" is yet a different aspect and it stroked us as a very promising vulnerability. Thus the idea of "posture spoofing" was born.

We define "posture spoofing" as an attack where a legitimate or illegitimate client spoofs "NAC posture credentials" in order get access unrestricted network access. The remainder of the paper is mostly dedicated to "which steps are necessary to realize a 'posture spoofing' and under which circumstances are these attacks feasible'.

Attacker's Definition

Different attack-vectors for "posture spoofing" are available for different types of attackers. Therefore we will first define these attackers.

Insider: An insider is a legitimate user of a NAC-protected network. The client has a working installation of the CTA and valid user/machine-credentials for the network. Additionally the inside attacker has the certificate of the ACS installed in its certificate store and if 802.1x is being used with client-certificates, this attacker has a valid certificate. The insider simply wants to bypass restrictions placed on his machine (e.g. no "leet tools" allowed and NAC checks list of installed programs).

Outsider: An outsider is not a legitimate user of the NAC-protected network and wants to get unrestricted access to the network. The outsider has no valid user/machine-credentials and no working CTA installation.

Attacker's Location

The logical location of the attacker with respect to the network and the enforcing device does not play a major role as the EAPoU or EAPo802.1x traffic always needs to be enabled.

Without adumbration of results presented later, the attack-vectors for "posture spoofing" with respect to "Attacker type" and "NAC flavour" are presented in the following table:

	Insider	**Outsider**
NAC-L2-802.1x	DLL/Plug-In replacement Scripting Interface CTA replacement[4]	None as to our current knowledge.
NAC-L2-IP	DLL/Plug-In replacement Scripting Interface CTA replacement	CTA replacement Scripting Interface
NACL-L3-IP	DLL/Plug-In replacement Scripting Interface CTA replacement	CTA replacement Scripting Interface

Table 5 Attack Vectors for posture spoofing

The crux with "posture spoofing" is that it has to be assumed that the attacker has no knowledge of the posture credentials which are required for "SPT healthy" in the actual setup. But, reading the official Cisco documentation it can be deduced, that the ACS actually requests credentials, thus promoting assisting an attacker with a classical "information disclosure" flaw which in itself seems harmless at first sight.

To sum it up, the analysis unveiled three design flaws:

1. Authorization without authentication.
2. Epimenides Paradox.
3. Information Disclosure through the ACS by providing the list of needed credentials.

[4] In the case of 802.1x a self-coded alternative client would need to also need to be a 802.1x supplicant with EAP-FAST support. This seems a bit of an "overkill" for that attack vector but it should nevertheless be mentioned for completeness sake.

Approaching NAC@ACK

After deciding to head at a “Posture Spoofing” attack a systematic approach was developed. The approach encompassed an attack-tree (which is depicted in Figure 14) and a set of techniques. From the techniques and the attack-tree “hacking steps” were derived and every step was first analyzed for its’ feasibility.

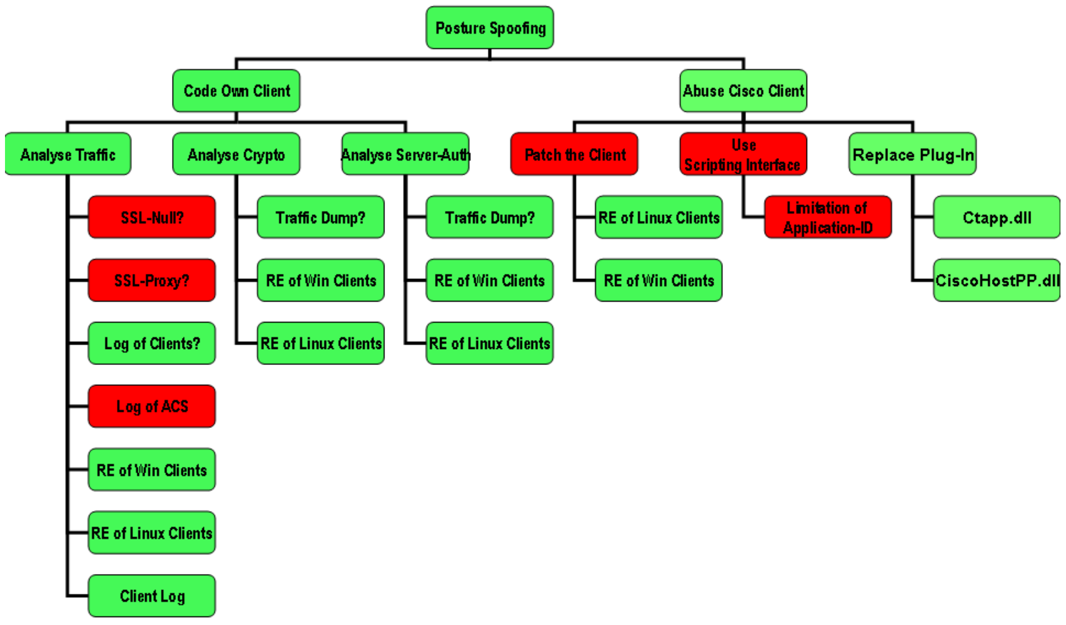


Figure 14 The Attack Tree. Green steps show “feasible” approaches whereas red steps have been shown to be not feasible or have no major benefit.The attack tree shown are not the first version but the latest version.

Brief primer on ‘hacking techniques’ for complex systems

The Cisco NAC framework is a proprietary closed-source complex system and hacking it requires a well defined set of tools and techniques. The process of hacking such complex systems is not comparable to the finding of Buffer Overflows the average reader might be more or less familiar with. FX did a very good job of describing a “complex system hack” with his “Analysing Complex Systems” presentation regarding Blackberry at the Blackhat-Europe 2006. A brief discussion of the techniques used shall be presented here before going into the results achieved. Some of these techniques yield results by themselves, others are either only useful in combination or serve as a staging process for a different technique.

Reverse Engineering: Reverse Engineering aims at uncovering the constructional elements of a product. In our case the product which needs reverse engineering is the binary version of CTA itself. The “tool-of choice” is the well-known disassembler IDAPro (thanks guys!)[5].

Packet Sniffing: As we are examining a networked system it strikes at obvious, that “packet sniffing” is one technique which should be taken into account. On a second look one can see, that packet sniffing is of limited use only, as the communication is encrypted and the packet-format is undisclosed. Nevertheless, Wireshark/Ethereal is your friend.

Packet Diffing: Understanding encrypted packets in an undisclosed format is quite challenging. One way to tackle the problem is by ‘diffing’ packets. By “diffing” we mean extracting common and differing parts of two packets. These differences hint at packet-bytes used for common tasks as e.g. “packet-code” [request/response] or “communication identifier” [comparable to a session-id].

[5] We will not go into the theory of RE – there are other people who are a lot better at that than we are and it is definitely out-of-scope for the paper. We assume you have a basic understanding of what RE is and how it works.

Multiple tools exist for diffing, even vim will serve as a simple binary diffing tool when combined with xxd.

Debugging / API-Monitoring: While reverse engineering or disassemblies provide the interested researcher with a snapshot of the program, it does not show the live process and the contents passed to the functions through the stack. That is where API-Monitoring and debugging have their say. Through attaching a debugger or api-monitor to the running process, it is possible to actually see the contents of the stack while the program is running. This is especially useful if the source code of the analysed program is not available. While we tested some different api-monitoring tool, we achieved the best results with "autodebug for windows"[6].

Built-in capabilities: Many software-products come with built in logging and/or debugging capabilities of various degrees. If these are available, they usually prove to be a very good source of information and should be used extensivly.

RTFM: Again this should be self-explanatory – reading all available documentation should be one of the first steps in any analysis. The original documentation may vary in quality but it usually contains many useful hints. And it is often worth reading more than once as some details in the documentation may be evalutaed differently on the second read with more knowledge from other sources at hand.

Applying the techniques

The first big "want to have" was a clear-text version of the encrypted traffic. As the traffic is secured with SSL/TLS the first approaches were to either force SSL-Null or use an SSL-Proxy. The SSL-Null was discarded quite quickly as it would need tampering with the client and/or the ACS and at the beginning of the research we wanted to avoid that. SSL-Proxy was not feasible either, as the NAC implements "SSL over UDP" and we have no knowledge of a Proxy which is able to do that. In addition the Cisco CTA has the certificate of the ACS installed and it validates the certificate in the process of the PEAP negotiation, so tampering with the certificate (e.g. through an SSL-Proxy) would lead to a malfunction of the SSL negotiation and therefore NAC would not work at all. While reading the (quite comprehensive) Cisco documentation of the CTA we discovered the "ctalogd.exe" – the part of CTA which is responsible for logging and which is configured through the "ctalogd.ini"-file. Logging writes its entries into a plaintext file on the client and every message is assigned a log level. The ctalogd.ini allows for the filtering of the messages according to the log level and a log level of 15 means "log everything". A test with log level 15 revealed that even decrypted packet-payloads are dumped – this was validated with the text from a user notification.

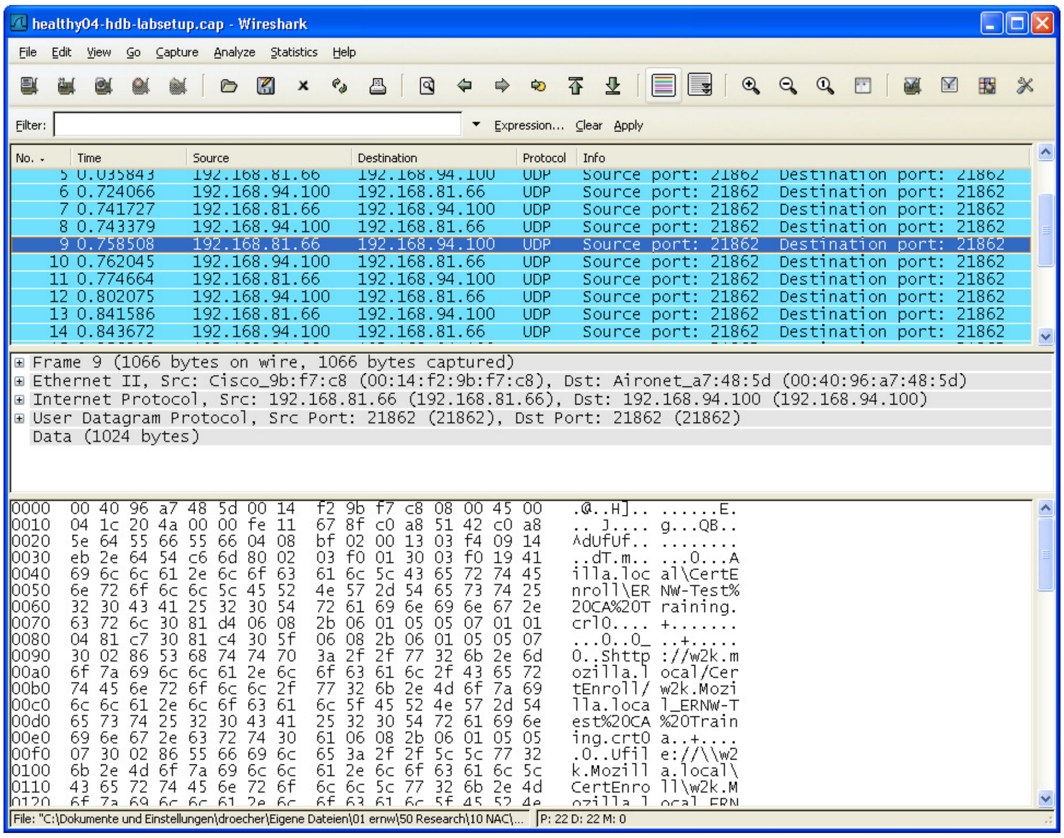
As to our knowledge the exact format of the packets is not documented anywhere (yet) [In February 2007 Cisco announced to publish the CTA as OpenSource but recalled that announcement later on[7]]– the best documentation for PEAPv1 is a Cisco Press book on wireless security[8] and there is an ietf-draf for "EAP over UDP" (expired August 2002)[9]. First hand source are (of course) the packet-dumps themselves and so we developed a test suite with multiple different policies for packet-capturing. Figure 15 shows a sample packet capture.

[6] Autodebug is available through http://www.autodebug.com/

[7] http://slashdot.org/article.pl?sid=07/02/08/2327220&from=rss

[8] Krishna Sankar et. al., „Cisco Wireless Lan Security" ISBN-13 978-1-58705-154-8, Nov. 2004, Ciscopress.

[9] http://folk.uio.no/paalee/publications/draft-engelstad-pana-eap-over-udp-00.txt


healthy04-hdb-labsetup.cap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply

```
No. -  Time       Source          Destination     Protocol Info
    5 0.035843   192.168.81.66   192.168.94.100  UDP      Source port: 21862  Destination port: 21862
    6 0.724066   192.168.94.100  192.168.81.66   UDP      Source port: 21862  Destination port: 21862
    7 0.741727   192.168.81.66   192.168.94.100  UDP      Source port: 21862  Destination port: 21862
    8 0.743379   192.168.94.100  192.168.81.66   UDP      Source port: 21862  Destination port: 21862
    9 0.758508   192.168.81.66   192.168.94.100  UDP      Source port: 21862  Destination port: 21862
   10 0.762045   192.168.94.100  192.168.81.66   UDP      Source port: 21862  Destination port: 21862
   11 0.774664   192.168.81.66   192.168.94.100  UDP      Source port: 21862  Destination port: 21862
   12 0.802075   192.168.94.100  192.168.81.66   UDP      Source port: 21862  Destination port: 21862
   13 0.841586   192.168.81.66   192.168.94.100  UDP      Source port: 21862  Destination port: 21862
   14 0.843672   192.168.94.100  192.168.81.66   UDP      Source port: 21862  Destination port: 21862
```

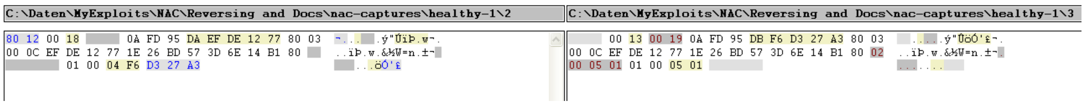
```
⊞ Frame 9 (1066 bytes on wire, 1066 bytes captured)
⊞ Ethernet II, Src: Cisco_9b:f7:c8 (00:14:f2:9b:f7:c8), Dst: Aironet_a7:48:5d (00:40:96:a7:48:5d)
⊞ Internet Protocol, Src: 192.168.81.66 (192.168.81.66), Dst: 192.168.94.100 (192.168.94.100)
⊞ User Datagram Protocol, Src Port: 21862 (21862), Dst Port: 21862 (21862)
  Data (1024 bytes)
```

```
0000  00 40 96 a7 48 5d 00 14  f2 9b f7 c8 08 00 45 00   .@..H].. ......E.
0010  04 1c 20 4a 00 00 fe 11  67 8f c0 a8 51 42 c0 a8   .. J.... g...QB..
0020  5e 64 55 66 55 66 04 08  bf 02 00 13 03 f4 09 14   ^dUfUf.. ........
0030  eb 2e 64 54 c6 6d 80 02  03 f0 01 30 03 f0 19 41   ..dT.m.. ...0...A
0040  69 6c 6c 61 2e 6c 6f 63  61 6c 5c 43 65 72 74 45   illa.loc al\CertE
0050  6e 72 6f 6c 6c 5c 45 52  4e 57 2d 54 65 73 74 25   nroll\ER NW-Test%
0060  32 30 43 41 25 32 30 54  72 61 69 6e 69 6e 67 2e   20CA%20T raining.
0070  63 72 6c 30 81 d4 06 08  2b 06 01 05 05 07 01 01   crl0.... +.......
0080  04 81 c7 30 81 c4 30 5f  06 08 2b 06 01 05 05 07   ...0..0_ ..+.....
0090  30 02 86 53 68 74 74 70  3a 2f 2f 77 32 6b 2e 6d   0..Shttp ://w2k.m
00a0  6f 7a 69 6c 6c 61 2e 6c  6f 63 61 6c 2f 43 65 72   ozilla.l ocal/Cer
00b0  74 45 6e 72 6f 6c 6c 2f  77 32 6b 2e 4d 6f 7a 69   tEnroll/ w2k.Mozi
00c0  6c 6c 61 2e 6c 6f 63 61  6c 5f 45 52 4e 57 2d 54   lla.loca l_ERNW-T
00d0  65 73 74 25 32 30 43 41  25 32 30 54 72 61 69 6e   est%20CA %20Train
00e0  69 6e 67 2e 63 72 74 30  61 06 08 2b 06 01 05 05   ing.crt0 a..+....
00f0  07 30 02 86 55 66 69 6c  65 3a 2f 2f 5c 5c 77 32   .0..Ufil e://\\w2
0100  6b 2e 4d 6f 7a 69 6c 6c  61 2e 6c 6f 63 61 6c 5c   k.Mozill a.local\
0110  43 65 72 74 45 6e 72 6f  6c 6c 5c 77 32 6b 2e 4d   CertEnro ll\w2k.M
0120  6f 7a 69 6c 6c 61 2e 6c  6f 63 61 6c 5f 45 52 4e   ozilla.l ocal_ERN
```

```
File: "C:\Dokumente und Einstellungen\droecher\Eigene Dateien\01 ernw\50 Research\10 NAC\...   P: 22 D: 22 M: 0
```

Figure 15 A sample packet capture. The payload shows the ACS certificate as part of the TLS establishment.

For further packet-analysis binary differences of the packet-payloads were taken. These “diffs” revealed bytes within the packets which are unchanged between “request” and “response” or between packets from the same source (e.g. all “client-packets” or all “ACS-packets”). A sample “diff” is shown in Figure 16.



```
C:\Daten\MyExploits\NAC\Reversing and Docs\nac-captures\healthy-1\2
80 12 00 18       0A FD 95 DA EF DE 12 77 80 03   ¬.. ..ýÚïÞ.w¬.
00 0C EF DE 12 77 1E 26 BD 57 3D 6E 14 B1 80      ..ïÞ.w.&½W=n.±¬
            01 00 04 F6 D3 27 A3                  ...öÓ'£

C:\Daten\MyExploits\NAC\Reversing and Docs\nac-captures\healthy-1\3
      00 13 00 19 0A FD 95 DB F6 D3 27 A3 80 03   .....ýÛöÓ'£¬.
00 0C EF DE 12 77 1E 26 BD 57 3D 6E 14 B1 80 02   ..ïÞ.w.&½W=n.±¬.
00 05 01 01 00 05 01                              .....
```

Figure 16 Sample packet-diffing

The ctalogd.ini file with log level 15 enabled:

```
[main]
EnableLog=1
LogDir=c:\temp
[LogLevel]
PADaemon=15
NetTrans=15
PAPlugin=15
[…]
```

Logging enabled

Log level "15" – log everything possible.

Excerpt from a CTA logfile:

```
65     16:23:13.343          04/26/2006        Sev=PktDump/13          CTAVSTLV/0x64300016
Request message dump:
000700D100000009800200C900000000900010010000100080000000000000000000000010000200
0800000000000000009000100A9000700A14865727A6C696368656E20476C7565636B77756E73
6368202D20496872205043206B6F6E6E7465206572666F6C677265696368206175746865E746
966697A6965727420776572646656E20756E6420656E74737072696368742064657220536563757
26974792050696F6C6963792E20496872652049E65747A7765726B7A7567616E67207769726420
6E696368742065696E676573636872E46E6B7421800300020001

66     16:23:13.359          04/26/2006        Sev=Info/4        PAPlugin/0x63200001
Application Posture Result = Healthy

67     16:23:13.359          04/26/2006        Sev=PktDump/13
Response message dump: 800300020001

68     16:23:13.359          04/26/2006        Sev=Debug/7       CT
EapHandlePacket exit

[...snipped...]

70     16:23:13.359          04/26/2006        Sev=Info/4        PAPlugin/0x63200002
System Posture Result = Healthy

71     16:23:13.359          04/26/2006        Sev=Warning/2   PAPlugin/0xA3200012
CTAPP received UserMsg Notification: Content = Herzlichen Glueckwunsch - Ihr PC konnte
erfolgreich authentifiziert werden und entspricht der Security Piolicy. Ihre Netzwerkzugang wird
nicht eingeschränkt!
```

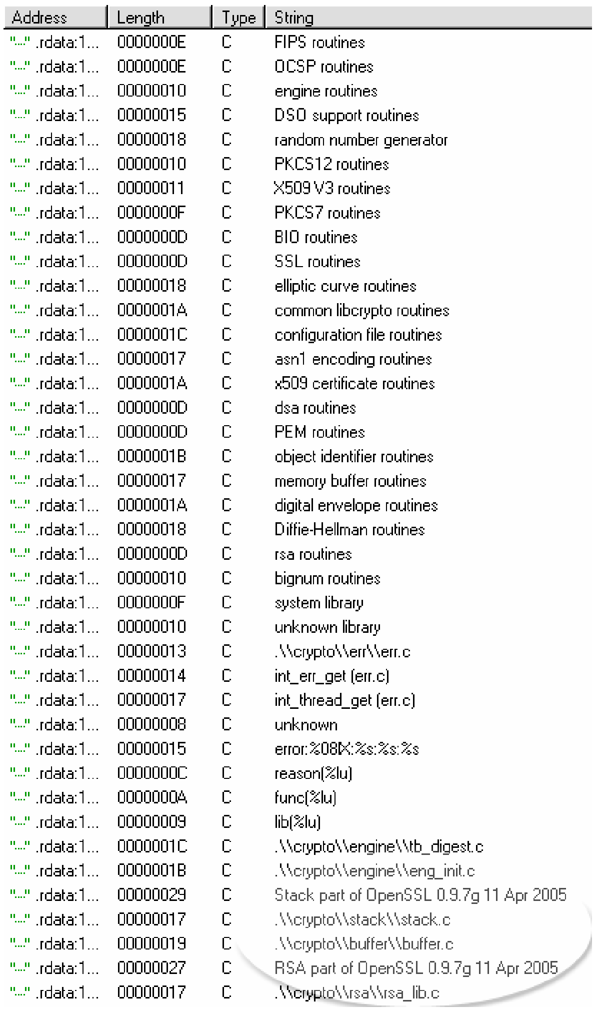
User Notification:
"Herzlichen …"

Convert to Hex:
%48%65%72%7a%6c%69
%63%68%65%6e%20

Reverse Engineering & Function Hooking Techniques

In order to get a better understanding of the inner workings of the CTA reversing techniques were applied. While the Windows version of CTA is compiled using some optimization methods, the Linux version is not optimized and so the focus of the reversing was first set to the Linux version. Figure 17 shows that the crypto within the CTA 2.0 for Linux is built around OpenSSL, meaning that it is possible to use existing crypto-libraries in coding an “alternative” NAC-client.

Besides revealing which (standard) crypto is being used, the reversing of the Linux client also sheds some light on the process-flow within the CTA itself and the core functions within the CTA. OpenSSL 0.9.7g is being used, which has known vulnerabilities[10] - attacking the client through OpenSSL was not pursued up to now but may proof to be a successful attack vector, but we would not consider this hacking of the NAC framework.



Address	Length	Type	String
"..." .rdata:1...	0000000E	C	FIPS routines
"..." .rdata:1...	0000000E	C	OCSP routines
"..." .rdata:1...	00000010	C	engine routines
"..." .rdata:1...	00000015	C	DSO support routines
"..." .rdata:1...	00000018	C	random number generator
"..." .rdata:1...	00000010	C	PKCS12 routines
"..." .rdata:1...	00000011	C	X509 V3 routines
"..." .rdata:1...	0000000F	C	PKCS7 routines
"..." .rdata:1...	0000000D	C	BIO routines
"..." .rdata:1...	0000000D	C	SSL routines
"..." .rdata:1...	00000018	C	elliptic curve routines
"..." .rdata:1...	0000001A	C	common libcrypto routines
"..." .rdata:1...	0000001C	C	configuration file routines
"..." .rdata:1...	00000017	C	asn1 encoding routines
"..." .rdata:1...	0000001A	C	x509 certificate routines
"..." .rdata:1...	0000000D	C	dsa routines
"..." .rdata:1...	0000000D	C	PEM routines
"..." .rdata:1...	0000001B	C	object identifier routines
"..." .rdata:1...	00000017	C	memory buffer routines
"..." .rdata:1...	0000001A	C	digital envelope routines
"..." .rdata:1...	00000018	C	Diffie-Hellman routines
"..." .rdata:1...	0000000D	C	rsa routines
"..." .rdata:1...	00000010	C	bignum routines
"..." .rdata:1...	0000000F	C	system library
"..." .rdata:1...	00000010	C	unknown library
"..." .rdata:1...	00000013	C	.\\crypto\\err\\err.c
"..." .rdata:1...	00000014	C	int_err_get (err.c)
"..." .rdata:1...	00000017	C	int_thread_get (err.c)
"..." .rdata:1...	00000008	C	unknown
"..." .rdata:1...	00000015	C	error:%08lX:%s:%s:%s
"..." .rdata:1...	0000000C	C	reason(%lu)
"..." .rdata:1...	0000000A	C	func(%lu)
"..." .rdata:1...	00000009	C	lib(%lu)
"..." .rdata:1...	0000001C	C	.\\crypto\\engine\\tb_digest.c
"..." .rdata:1...	0000001B	C	.\\crypto\\engine\\eng_init.c
"..." .rdata:1...	00000029	C	Stack part of OpenSSL 0.9.7g 11 Apr 2005
"..." .rdata:1...	00000017	C	.\\crypto\\stack\\stack.c
"..." .rdata:1...	00000019	C	.\\crypto\\buffer\\buffer.c
"..." .rdata:1...	00000027	C	RSA part of OpenSSL 0.9.7g 11 Apr 2005
"..." .rdata:1...	00000017	C	.\\crypto\\rsa\\rsa_lib.c

Figure 17 CTA for Linux is built on OpenSSL libraries.

The two most promising attack-vectors are “plug-in replacement” and “alternative client”. Even though it would be very easy to implement, the “scripting interface” is deemed less feasible because of limitations placed on the “application id” and “vendor name” for posture-credentials generated through scripts. As mentioned earlier the vendor id for scripts is always “Cisco” and the application id has to be from the “private” range. As to now it is not clear if these restrictions can be circumvented in any way without replacing the scripting-interface-dll.

For the “alternative client” it is essential to understand how the payload is built and transported. The disassembly shows that the core function of the CTA for that purpose is “TlvHandlePacket” (Figure 18). A disassembly of the components of the CTA on its own reveals core functionality and

[10] Chek BID 20246, 15071 and 20249

core functions but it can not trace an application while it is running. The possibility to trace an application at runtime and see calls to functions and contents of stacks and buffers is pivotal to understand what is actually going on within the application and for that reason "function hooking/api-monitoring" was used. The results from the disassembly were used to identify "interesting" functions for hooking into.

Figure 19 shows a call to "EapTlvHandlePacket" with posture data passed to that function. The buffer in contains a list of all installed Microsoft hotfixes concatenated with a pipe, the hostname of the machine (vm-xp1-nocta), the name of the logged-in user (droecher) and the operating system (Windows XP SP2). This result is somewhat surprising, as the policy defined on the ACS during the test-run had no rule regarding installed patches/hotfixes. A possible conclusion is, that the CTA always submits all available posture-credentials to the ACS and that the ACS then only evaluates the credentials which are configured in a policy. This seems to contradict the earlier stated design flaw "information disclosure".

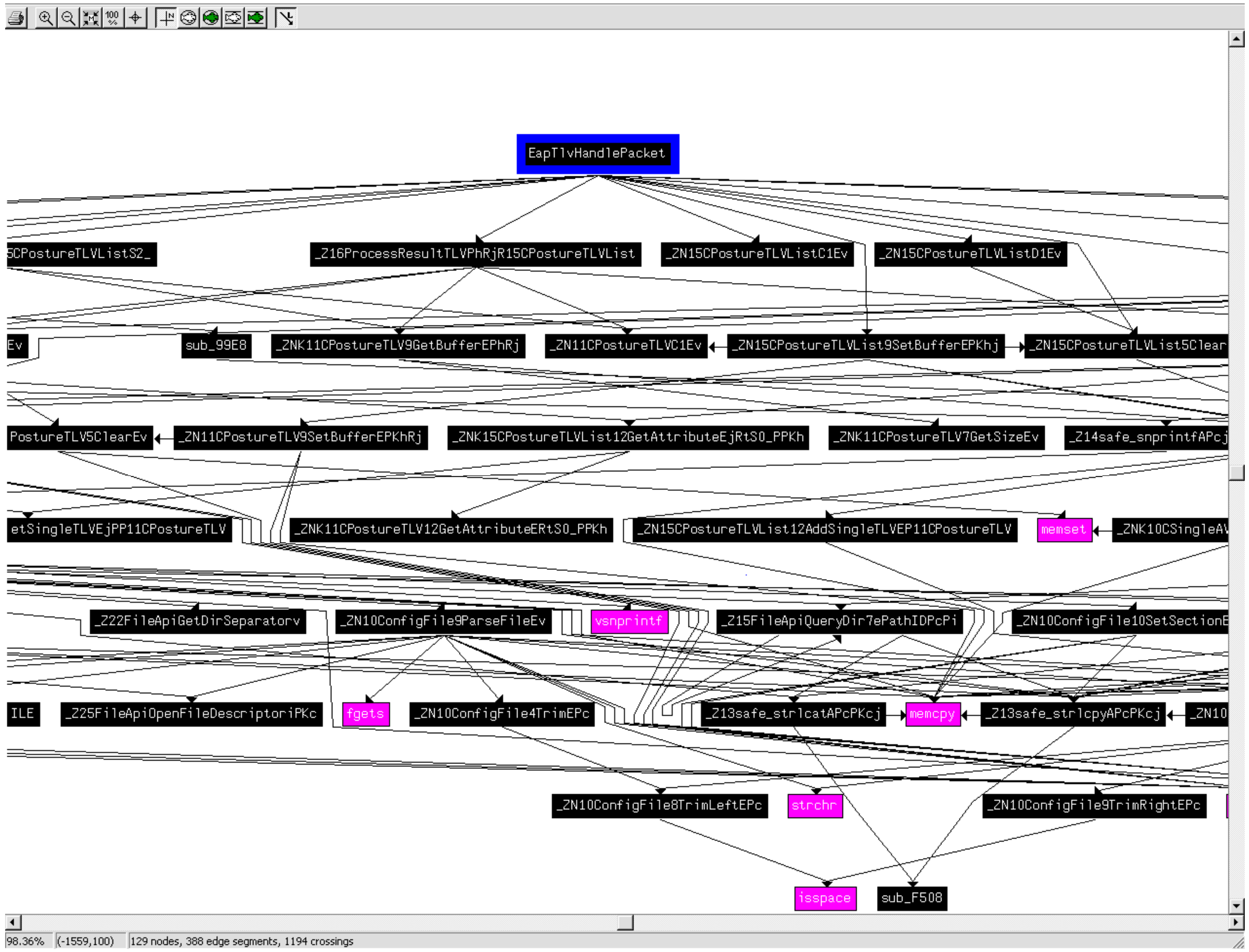
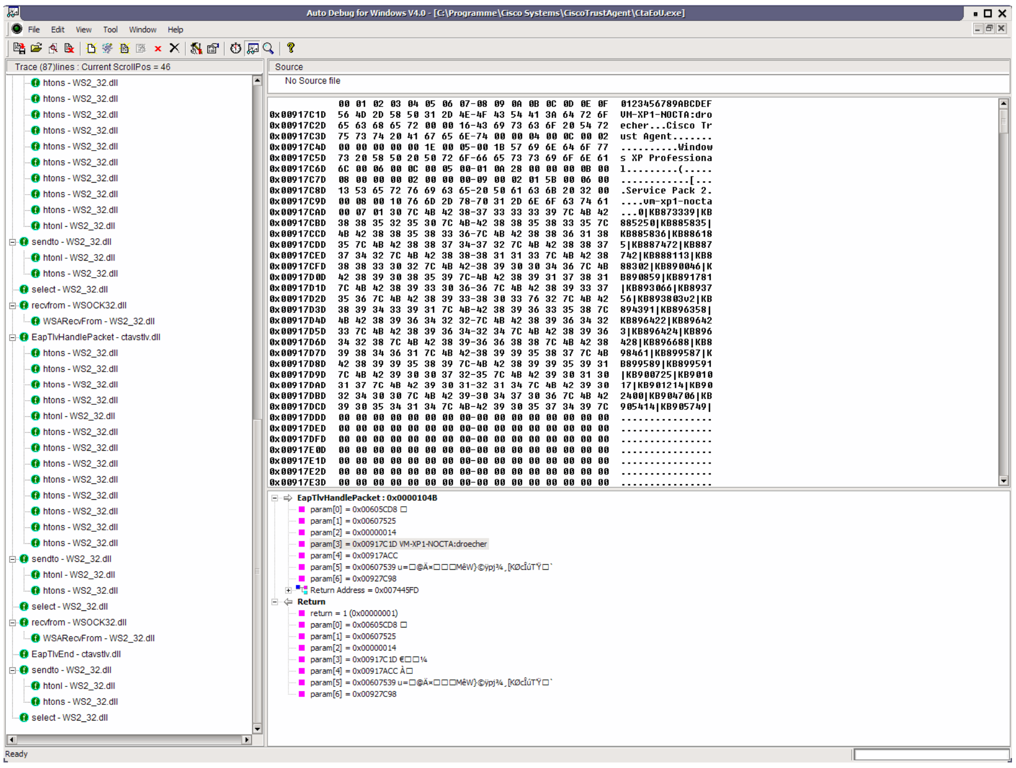

EapTlvHandlePacket
5CPostureTLVListS2_
_Z16ProcessResultTLVPhRjR15CPostureTLVList
_ZN15CPostureTLVListC1Ev
_ZN15CPostureTLVListD1Ev
Ev
sub_99E8
_ZNK11CPostureTLV9GetBufferEPhRj
_ZN11CPostureTLVC1Ev
_ZN15CPostureTLVList9SetBufferEPKhj
_ZN15CPostureTLVList5Clear
PostureTLV5ClearEv
_ZN11CPostureTLV9SetBufferEPKhRj
_ZNK15CPostureTLVList12GetAttributeEjRtS0_PPKh
_ZNK11CPostureTLV7GetSizeEv
_Z14safe_snprintfAPcj
etSingleTLVEjPP11CPostureTLV
_ZNK11CPostureTLV12GetAttributeERtS0_PPKh
_ZN15CPostureTLVList12AddSingleTLVEP11CPostureTLV
memset
_ZNK10CSingleA
_Z22FileApiGetDirSeparatorv
_ZN10ConfigFile9ParseFileEv
vsnprintf
_Z15FileApiQueryDir7ePathIDPcPi
_ZN10ConfigFile10SetSection
ILE
_Z25FileApiOpenFileDescriptoriPKc
fgets
_ZN10ConfigFile4TrimEPc
_Z13safe_strlcatAPcPKcj
memcpy
_Z13safe_strlcpyAPcPKcj
_ZN10
_ZN10ConfigFile8TrimLeftEPc
strchr
_ZN10ConfigFile9TrimRightEPc
isspace
sub_F508
98.36%
(-1559,100)
129 nodes, 388 edge segments, 1194 crossings

Figure 18 Core function „EapTlvHandlePacket“ from reversing of the CTA



```
Auto Debug for Windows V4.0 - [C:\Programme\Cisco Systems\CiscoTrustAgent\CtaEoU.exe]

Trace (87)lines : Current ScrollPos = 46
    htons - WS2_32.dll
    htons - WS2_32.dll
    htons - WS2_32.dll
    htons - WS2_32.dll
    htons - WS2_32.dll
    htons - WS2_32.dll
    htons - WS2_32.dll
    htons - WS2_32.dll
    htons - WS2_32.dll
    htonl - WS2_32.dll
sendto - WS2_32.dll
    htonl - WS2_32.dll
    htons - WS2_32.dll
select - WS2_32.dll
recvfrom - WSOCK32.dll
    WSARecvFrom - WS2_32.dll
EapTlvHandlePacket - ctavstlv.dll
    htons - WS2_32.dll
    htons - WS2_32.dll
    htons - WS2_32.dll
    htons - WS2_32.dll
    htonl - WS2_32.dll
    htons - WS2_32.dll
    htons - WS2_32.dll
    htons - WS2_32.dll
    htons - WS2_32.dll
    htons - WS2_32.dll
    htons - WS2_32.dll
    htons - WS2_32.dll
    htons - WS2_32.dll
sendto - WS2_32.dll
    htonl - WS2_32.dll
    htons - WS2_32.dll
select - WS2_32.dll
recvfrom - WSOCK32.dll
    WSARecvFrom - WS2_32.dll
EapTlvEnd - ctavstlv.dll
sendto - WS2_32.dll
    htonl - WS2_32.dll
    htons - WS2_32.dll
select - WS2_32.dll
Ready

Source
  No Source file

            00 01 02 03 04 05 06 07-08 09 0A 0B 0C 0D 0E 0F  0123456789ABCDEF
0x00917C1D  56 4D 2D 58 50 31 2D 4E-4F 43 54 41 3A 64 72 6F  VM-XP1-NOCTA:dro
0x00917C2D  65 63 68 65 72 00 00 16-43 69 73 63 6F 20 54 72  echer...Cisco Tr
0x00917C3D  75 73 74 20 41 67 65 6E-74 00 00 04 00 0C 00 02  ust Agent.......
0x00917C4D  00 00 00 00 00 1E 00 05-00 1B 57 69 6E 64 6F 77  ..........Window
0x00917C5D  73 20 58 50 20 50 72 6F-66 65 73 73 69 6F 6E 61  s XP Professiona
0x00917C6D  6C 00 06 00 0C 00 05 00-01 0A 28 00 00 00 0B 00  l.........(.....
0x00917C7D  08 00 00 00 02 00 00 00-09 00 02 01 5B 00 06 00  ............[...
0x00917C8D  13 53 65 72 76 69 63 65-20 50 61 63 6B 20 32 00  .Service Pack 2.
0x00917C9D  00 08 00 10 76 6D 2D 78-70 31 2D 6E 6F 63 74 61  ....vm-xp1-nocta
0x00917CAD  00 07 01 30 7C 4B 42 38-37 33 33 33 39 7C 4B 42  ...0|KB873339|KB
0x00917CBD  38 38 35 32 35 30 7C 4B-42 38 38 35 38 33 35 7C  885250|KB885835|
0x00917CCD  4B 42 38 38 35 38 33 36-7C 4B 42 38 38 36 31 38  KB885836|KB88618
0x00917CDD  35 7C 4B 42 38 38 37 34-37 32 7C 4B 42 38 38 37  5|KB887472|KB887
0x00917CED  37 34 32 7C 4B 42 38 38-38 31 31 33 7C 4B 42 38  742|KB888113|KB8
0x00917CFD  38 38 33 30 32 7C 4B 42-38 39 30 30 34 36 7C 4B  88302|KB890046|K
0x00917D0D  42 38 39 30 38 35 39 7C-4B 42 38 39 31 37 38 31  B890859|KB891781
0x00917D1D  7C 4B 42 38 39 33 30 36-36 7C 4B 42 38 39 33 37  |KB893066|KB8937
0x00917D2D  35 36 7C 4B 42 38 39 33-38 30 33 76 32 7C 4B 42  56|KB893803v2|KB
0x00917D3D  38 39 34 33 39 31 7C 4B-42 38 39 36 33 35 38 7C  894391|KB896358|
0x00917D4D  4B 42 38 39 36 34 32 32-7C 4B 42 38 39 36 34 32  KB896422|KB89642
0x00917D5D  33 7C 4B 42 38 39 36 34-32 34 7C 4B 42 38 39 36  3|KB896424|KB896
0x00917D6D  34 32 38 7C 4B 42 38 39-36 36 38 38 7C 4B 42 38  428|KB896688|KB8
0x00917D7D  39 38 34 36 31 7C 4B 42-38 39 39 35 38 37 7C 4B  98461|KB899587|K
0x00917D8D  42 38 39 39 35 38 39 7C-4B 42 38 39 39 35 39 31  B899589|KB899591
0x00917D9D  7C 4B 42 39 30 30 37 32-35 7C 4B 42 39 30 31 30  |KB900725|KB9010
0x00917DAD  31 37 7C 4B 42 39 30 31-32 31 34 7C 4B 42 39 30  17|KB901214|KB90
0x00917DBD  32 34 30 30 7C 4B 42 39-30 34 37 30 36 7C 4B 42  2400|KB904706|KB
0x00917DCD  39 30 35 34 31 34 7C 4B-42 39 30 35 37 34 39 7C  905414|KB905749|
0x00917DDD  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0x00917DED  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0x00917DFD  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0x00917E0D  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0x00917E1D  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0x00917E2D  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0x00917E3D  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................

EapTlvHandlePacket : 0x0000104B
    param[0] = 0x00605CD8 □
    param[1] = 0x00607525
    param[2] = 0x00000014
    param[3] = 0x00917C1D VM-XP1-NOCTA:droecher
    param[4] = 0x00917ACC
    param[5] = 0x00607539 u=□@Ä¤□□□MêW}©ÿpj¾¸[KØcÎúTŸ□`
    param[6] = 0x00927C98
    Return Address = 0x007445FD
Return
    return = 1 (0x00000001)
    param[0] = 0x00605CD8 □
    param[1] = 0x00607525
    param[2] = 0x00000014
    param[3] = 0x00917C1D €□□¼
    param[4] = 0x00917ACC À□
    param[5] = 0x00607539 u=□@Ä¤□□□MêW}©ÿpj¾¸[KØcÎúTŸ□`
    param[6] = 0x00927C98
```

Figure 19 Function hooking reveals parameters passed to functions.

But even more promising than "alternative client" is the approach of "plug-in" replacement. If you recall the structure of the CTA (Figure 3) it becomes clear, that the (probably most commonly used) posture-data for "PA" and "Host" is collected using two corresponding plug-ins. The difference between a plug-in and a self-written script is that the script writes the posture-data to a file and invokes "ctasi.exe" whereas a plug-in hands the posture-data over to the CTA using exported DLL-calls. Additionally the "vendor id" and "application id" restrictions do not apply to plug-ins but only to scripts. So by analyzing different plug-ins for common exported functions it is possible to identify these functions (for an example of the disassembly see Figure 20) and thereby reconstruct the plug-in interface not publicly disclosed by Cisco. Three common functions have been identified in the plug-in feature of the CTA and the disassembly shows their functionality and structure (Figure 21Disassembly of the public plug-in functionsFigure 21 shows parts of the disassemblies of the core functions of the plug-in interface).

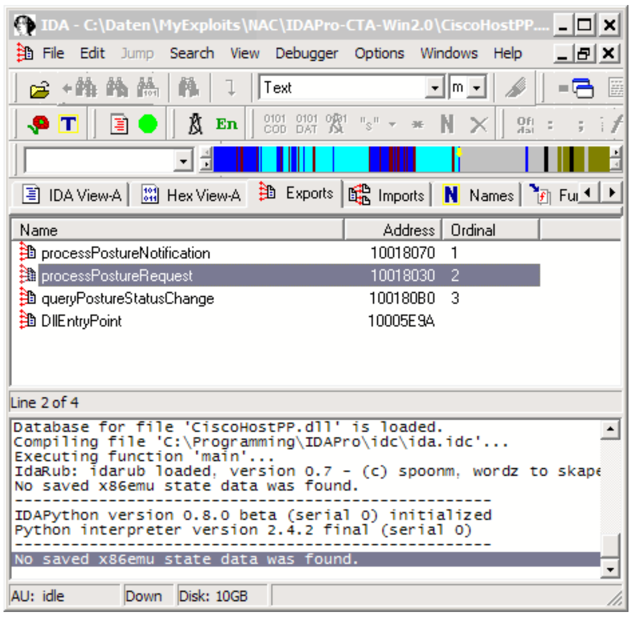
The conclusion

Using the readily available documentation and the information provided above, the development of plug-ins for the CTA is not too difficult for a skilled programmer. And instead of just developing "a plug-in", replacement-plug-ins for existing plug-ins can be developed (especially for the PA and Host plug-in – simply "replace" them with self-written plug-ins), therefore rendering Cisco NAC more or less useless in all "insider"-scenarios[11].

For "outsider-scenarios" the situation is somewhat more difficult. As the outsider is not in the possession of the ACS certificate and has no valid credentials, a simple "replace plug-in dlls" will not do. Even if the outsider would install the CTA and replace the plug-ins he would still be lacking the ACS certificate without which the CTA will fail to establish the PEAP session.

The outsider needs a complete "new" replacement-client. A key feature of this client (besides spoofing arbitrary posture credentials) would be to accept the ACS certificate as valid and use it for PEAP without it being installed in a local certificate-store beforehand. But again this "alternative client approach" is quite definitely possible without too much trouble as the modular CTA allows to "reuse" part of the program, as for example the PEAP-libraries provided by Cisco with the CTA

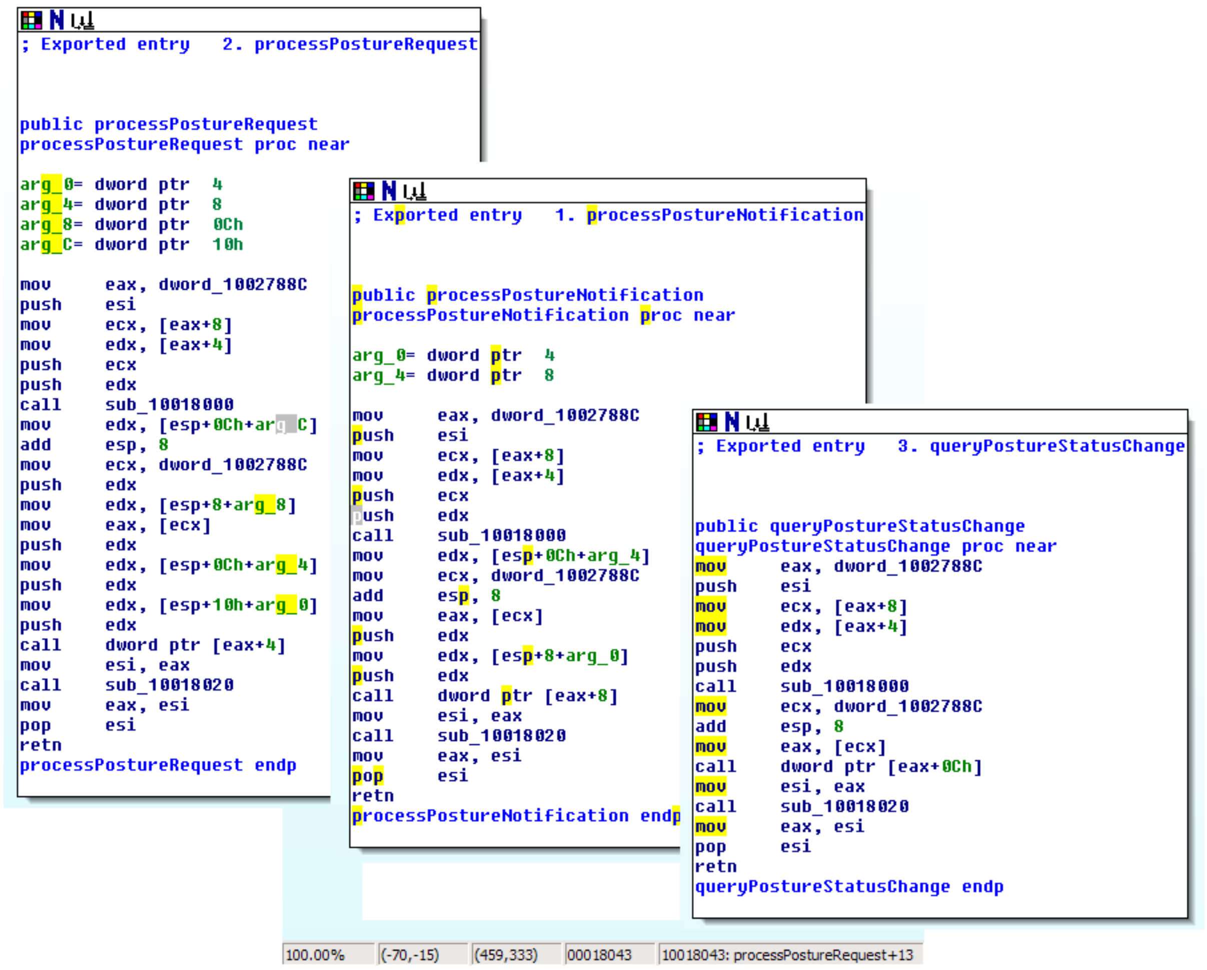
[11] This is a somewhat provocative statement. When talking about "mitigation in the next section, we will discuss how NAC can be useful and how it could be improved.


IDA - C:\Daten\MyExploits\NAC\IDAPro-CTA-Win2.0\CiscoHostPP....
File Edit Jump Search View Debugger Options Windows Help
Text
IDA View-A
Hex View-A
Exports
Imports
Names
Line 2 of 4
AU: idle
Down
Disk: 10GB

Name	Address	Ordinal
processPostureNotification	10018070	1
processPostureRequest	10018030	2
queryPostureStatusChange	100180B0	3
DllEntryPoint	10005E9A	

```
Database for file 'CiscoHostPP.dll' is loaded.
Compiling file 'C:\Programming\IDAPro\idc\ida.idc'...
Executing function 'main'...
IdaRub: idarub loaded, version 0.7 - (c) spoonm, wordz to skape
No saved x86emu state data was found.
--------------------------------------------------------
IDAPython version 0.8.0 beta (serial 0) initialized
Python interpreter version 2.4.2 final (serial 0)
--------------------------------------------------------
No saved x86emu state data was found.
```

Figure 20 Sample disassembly of CiscoHostPP.dll plug-in showing the exported functions.



```
; Exported entry   2. processPostureRequest


public processPostureRequest
processPostureRequest proc near

arg_0= dword ptr  4
arg_4= dword ptr  8
arg_8= dword ptr  0Ch
arg_C= dword ptr  10h

mov     eax, dword_1002788C
push    esi
mov     ecx, [eax+8]
mov     edx, [eax+4]
push    ecx
push    edx
call    sub_10018000
mov     edx, [esp+0Ch+arg_C]
add     esp, 8
mov     ecx, dword_1002788C
push    edx
mov     edx, [esp+8+arg_8]
mov     eax, [ecx]
push    edx
mov     edx, [esp+0Ch+arg_4]
push    edx
mov     edx, [esp+10h+arg_0]
push    edx
call    dword ptr [eax+4]
mov     esi, eax
call    sub_10018020
mov     eax, esi
pop     esi
retn
processPostureRequest endp
```

```
; Exported entry   1. processPostureNotification


public processPostureNotification
processPostureNotification proc near

arg_0= dword ptr  4
arg_4= dword ptr  8

mov     eax, dword_1002788C
push    esi
mov     ecx, [eax+8]
mov     edx, [eax+4]
push    ecx
push    edx
call    sub_10018000
mov     edx, [esp+0Ch+arg_4]
mov     ecx, dword_1002788C
add     esp, 8
mov     eax, [ecx]
push    edx
mov     edx, [esp+8+arg_0]
push    edx
call    dword ptr [eax+8]
mov     esi, eax
call    sub_10018020
mov     eax, esi
pop     esi
retn
processPostureNotification endp
```

```
; Exported entry   3. queryPostureStatusChange


public queryPostureStatusChange
queryPostureStatusChange proc near
mov     eax, dword_1002788C
push    esi
mov     ecx, [eax+8]
mov     edx, [eax+4]
push    ecx
push    edx
call    sub_10018000
mov     ecx, dword_1002788C
add     esp, 8
mov     eax, [ecx]
call    dword ptr [eax+0Ch]
mov     esi, eax
call    sub_10018020
mov     eax, esi
pop     esi
retn
queryPostureStatusChange endp
```

```
100.00%  (-70,-15)  (459,333)  00018043  10018043: processPostureRequest+13
```

Figure 21Disassembly of the public plug-in functions

.

Thoughts on Mitigation

If you want to use Cisco NAC, go ahead – but first define what you want to achieve and then think about which NAC setup will fit your goals best. Then calculate the costs, think again, evaluate different approaches or different solutions (Cisco NAC is not the only solution on the market), think again, do a risk-analysis, calculate again, rethink and then you might decide, that you still want to use it. If that is true for you, then go ahead. But back to business – that the current version of Cisco NAC is flawed has been extensively shown. The question is – who can do what to make the solution "secure"? As we are not talking about a "coding error" leading to a buffer overflow but about a fundamental design-flaw, a simple "patch the client" will not work.

There are two parties who can improve the security of the solution: Cisco (of course) and customers using Cisco NAC.

What Cisco can do

Cisco could improve the design of the NAC framework in order to render attacks from insider and outsider useless. Here are some thoughts on what could be done:

Code Signing: Code Signing[12] the plug-ins and running only signed plug-ins would defeat plug-in replacement attacks thereby defeating insider attacks. We can not judge the effort needed to implement code signing but we would heartily welcome seeing signed code in any security product.

Mandatory Authentication: Strong mandatory authentication would stop outsider attacks against the NAC framework. Adding authentication (mandatory or, in a first step, optional) should be possible without too much of a change as PEAP is being used and PEAP has built-in authentication. Preferably we would like to see smart-card based authentication against a central user-directory – but that may be wishing for too much. The reasons for not having authentication in that framework can only be business-related – Cisco knows that implementing NAC is already a major effort and does not want to put additional stress on its clients by making authentication mandatory.

What customers can do

The security of Cisco NAC highly depends on the overall setup/installation of the client and on the NAC flavor chosen.

Strong Authentication: Whenever possible 802.1x-based NAC should be implemented in order to add strong authentication to the authorization process. If 802.1x is not feasible, other means of strong authentication should be implemented. In RAS-VPN scenarios for example, where NAC-Layer3-IP is the only NAC-flavor available, clients should be subjected to strong authentication on the VPN-device itself. The "strong authentication" mitigates threats posed by the "outside attacker".

Least Privilege: All attack-vectors for "inside attackers" have a common characteristic. They need "tampering" with the CTA installation. In case of "plug-in"-replacement the authentic plug-ins are being replaced by self-written plug-ins. Two possible mitigations against this attack would be to (a) enforce strict access-rights on the plug-in files which prevent replacement (this being a preventive measure) and (b) use some sort of "file altering monitor" to detect changed plug-in files. The second listed mitigation is the less preferable one as it does not prevent but only detect. In case of "alternative client" neither "file monitoring" nor "file access restrictions" are possible mitigations. But an alternative client would need to open a listening UDP port 28162 – which could be restricted by a local firewall to the original CTA.

CSA instead of CTA: In addition to the CTA Cisco also offers a host based IDS in the name of "Cisco Security Agent" which also includes the CTA and has its own CTA plug in. The CSA monitors the integrity of the CTA and will prevent illegitimate changes to the CTA. This will mitigate threats posed by the "inside attacker".

[12] A good introduction to code signing can be found here: http://msdn.microsoft.com/workshop/security/authcode/intro_authenticode.asp

List of Figures

References & Further Reading

General Cisco sites:

- http://www.cisco.com/go/nac
- http://www.cisco.com/go/sdn
- http://www.cisco.com/go/cta

Cisco CTA & NAC guides

- NAC deployment guide http://www.cisco.com/application/pdf/en/us/guest/netsol/ns617/c649/cdccont_0900aecd80417226.pdf
- CTA 2.1 Administrator Guide http://www.cisco.com/application/pdf/en/us/guest/products/ps5923/c2001/ccmigration_09186a00807ec2f7.pdf

Books

- "Cisco Network Admission Control, Volume I: NAC Framework Architecture and Design", ISBN-13: 978-1-58705-241-5;
- "Cisco Network Admission Control, Volume II: NAC Deployment and Troubleshooting", ISBN-13: 978-1-58705-225-5
- "Cisco Wireless LAN Security", ISBN-13: 978-1-58705-154-8

Acknowledgements

Without the support of some people all this research work would not have been possible – therefor we want to thank these persons:

- Enno Rey & Roland Fiege, ERNW GmbH, for providing us with lab-equipment and some spare time.
- Simon Rich, ERNW GmbH, for helping us with the coding of the client.
- Barbara Jantzen & Hannah Jantzen, for bearing with my absence.