

07/06/01
945 S. PTO

07-09-01

APPROV

Pocket No. VIV/0003.00

Please type a plus sign (+) inside this box

PTO/SB/16 (8-00)

Approved for use through 10/31/2002. OMB 0651-0032

U.S. Patent and Trademark Office, U.S. DEPARTMENT OF COMMERCE

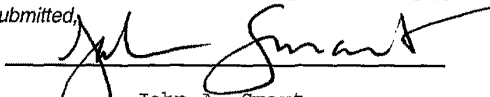
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number

1011 U.S. PTO
60/303653
07/06/01

PROVISIONAL APPLICATION FOR PATENT COVER SHEET

This is a request for filing a PROVISIONAL APPLICATION FOR PATENT under 37 CFR 1.53(c).

INVENTOR(S)					
Given Name (first and middle [if any])	Family Name or Surname	Residence (City and either State or Foreign Country)			
Gregor Paul	Freund	San Francisco, CA			
Keith Allen	Haycock	San Francisco, CA			
Conrad Kamaha'o	Herrmann	Oakland, CA			
<input type="checkbox"/> Additional inventors are being named on the _____ separately numbered sheets attached hereto					
TITLE OF THE INVENTION (280 characters max)					
System Providing Internet Access Management with Router-based Policy Enforcement					
Direct all correspondence to: CORRESPONDENCE ADDRESS					
<input checked="" type="checkbox"/> Customer Number	28653	<div style="border: 1px solid black; padding: 5px;">Place Customer Number Bar Code Label here</div>			
OR Type Customer Number here					
<input type="checkbox"/> Firm or Individual Name	John A. Smart				
Address					
708 Blossom Hill Rd., #201					
City	Los Gatos	State	CA	ZIP	95032-3503
Country	U.S.A.	Telephone	(408) 395-8819	Fax	(408) 490-2853
ENCLOSED APPLICATION PARTS (check all that apply)					
<input checked="" type="checkbox"/> Specification	Number of Pages	94	<input type="checkbox"/> CD(s), Number		
<input checked="" type="checkbox"/> Drawing(s)	Number of Sheets	(embedded)	<input checked="" type="checkbox"/> Other (specify)	Express Mail Certificate	
<input checked="" type="checkbox"/> Application Data Sheet	See 37 CFR 1.76				
METHOD OF PAYMENT OF FILING FEES FOR THIS PROVISIONAL APPLICATION FOR PATENT					
<input checked="" type="checkbox"/>	Applicant claims small entity status. See 37 CFR 1.27.				FILING FEE AMOUNT (\$)
<input checked="" type="checkbox"/>	A check or money order is enclosed to cover the filing fees				
<input type="checkbox"/>	The Commissioner is hereby authorized to charge filing fees or credit any overpayment to Deposit Account Number: _____				75.00
<input type="checkbox"/>	Payment by credit card. Form PTO-2038 is attached.				
The invention was made by an agency of the United States Government or under a contract with an agency of the United States Government.					
<input checked="" type="checkbox"/>	No.				
<input type="checkbox"/>	Yes, the name of the U.S. Government agency and the Government contract number are: _____				

Respectfully submitted,

SIGNATURE _____
TYPED or PRINTED NAME John A. Smart
TELEPHONE (408) 395-8819

Date 07/06/2001
REGISTRATION NO. 34,929
(if appropriate)
Docket Number: VIV/0003.00

USE ONLY FOR FILING A PROVISIONAL APPLICATION FOR PATENT

This collection of information is required by 37 CFR 1.51. The information is used by the public to file (and by the PTO to process) a provisional application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 8 hours to complete, including gathering, preparing, and submitting the complete provisional application to the PTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, Washington, D.C. 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Box Provisional Application, Assistant Commissioner for Patents, Washington, D.C. 20231.

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

Certificate under ~~37 CFR 1.10~~ of Mailing by "Express Mail"

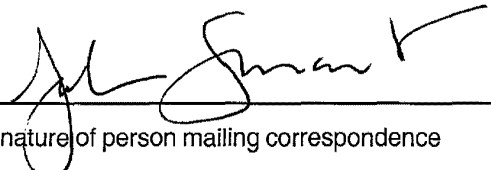
EF062711990US

"Express Mail" label number

July 6, 2001

Date of Deposit

I hereby certify that this correspondence is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231.



Signature of person mailing correspondence

John A. Smart

Typed or printed name of person mailing correspondence

Note: Each paper must have its own certificate of mailing by "Express Mail".

109060-6590009

FOR REVIEW

Application Data Sheet (ADS) Attachment
Docket No. VIV/0003.00

FOIA b 7 - C

INVENTOR INFORMATION

Inventor One Given Name:: Gregor
Family Name:: Freund
Postal Address Line One:: 26 Vulcan Stairways
City:: San Francisco
State:: CA
Zip:: 94114
Citizenship Country:: Germany

Inventor Two Given Name:: Keith
Family Name:: Haycock
City:: San Francisco
State:: CA
Citizenship Country:: US

Inventor Three Given Name:: Conrad
Family Name:: Herrmann
City:: Oakland
State:: CA
Citizenship Country:: US

CORRESPONDENCE INFORMATION

Correspondence Customer Number:: 28653

APPLICATION INFORMATION

Title Line One:: System Providing Internet Access
Title Line Two:: Management with Router-based Policy
Title Line Three:: Enforcement
Total Drawing Sheets:: 0
Formal Drawings?:: No
Application Type:: Utility
Docket Number:: VIV/0003.00

REPRESENTATIVE INFORMATION

Representative Customer Number:: 28653

I hereby certify that this correspondence is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service on the date indicated below and is addressed to Box Provisional Application, Assistant Commissioner for Patents, Washington, D.C. 20231.

Docket No. **VIV/0003.00**

"Express Mail" label number:

Date: **July 6, 2001**

By:


John A. Smart

PROVISIONAL PATENT APPLICATION

SYSTEM PROVIDING INTERNET ACCESS MANAGEMENT WITH ROUTER-BASED POLICY ENFORCEMENT

Inventors: GREGOR PAUL FREUND, a citizen of German residing in San Francisco, CA; KEITH ALLEN HAYCOCK, a citizen of The United States residing in San Francisco, CA; and CONRAD KAMAHA'O HERRMANN, a citizen of The United States residing in Oakland, CA.

Assignee: Zone Labs, Inc.

John A. Smart
Reg. No. 34,929

708 Blossom Hill Rd., #201
Los Gatos, CA 95032-3503
(408) 395-8819; (408) 490-2853 FAX

109020-636003

SYSTEM PROVIDING INTERNET ACCESS MANAGEMENT WITH ROUTER-BASED
POLICY ENFORCEMENT

COPYRIGHT NOTICE

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

Internet-connected system with computer-implemented methodology

The present invention may be implemented in a data processing environment (e.g., client/server database system) including one or more computer systems having Internet connectivity, such as one or more client computers (e.g., workstation, terminal, desktop PC, or the like) running client software, connected over a network (e.g., 10/100 Base T/Ethernet) to a server computer running back-end server software. A suitable database system for implementing the present invention is described in the following commonly-owned U.S. applications:

Docket No. VIV/0001.01
In re: Gregor Freund
Filing Date: 05/06/1997
Serial No.: 08/851,777
Patent No.: 5,987,611
Issue Date: 11/16/1999
Title: System and Methodology for Managing Internet Access on a Per Application Basis for Client Computers Connected to the Internet (as amended)

The disclosures of the foregoing are hereby incorporated by reference in their entirety, including any appendices or attachments thereof, for all purposes.

FOIA b 7 - D

Attachment #1:
Internet Access Management with Router-based Policy Enforcement (23 pages)

5

10

TABLE OF CONTENTS

Attachment #2:
Source code attachment (66 pages)

Zone Labs



Zone Labs -

Provisional Patent Application for System Providing Internet Access
Management with Router-based Policy Enforcement

FOR CONFIDENTIAL

Zone Labs

Table of Contents

1	Summary	3
1.1	Key Characteristics:	4
1.2	Compelling Business factors:	4
2	Functional Overview	5
2.1	Core Features – Internet Access Enforcement:	5
	Figure 2: Router Setup screen	7
2.2	ZAP enforcement Overview	8
2.3	Optional ZAP Version Control Overview	8
2.4	Optional Licensing Key Management Overview	8
2.5	Optional Virus Checking Overview	9
2.6	Sandbox Server Overview	9
2.7	Future feature enhancements	9
3	Procedural Overview	10
3.1	Implementation	10
3.2	Client Monitoring Protocol Overview	11
4	Technical details	13
4.1	ZAP enforcement Router administration panel	13
4.2	Sandbox server	14
4.3	Client monitoring protocol (CMP)	15
4.3.1	<i>Client Hello</i>	15
4.3.2	<i>Router challenge</i>	16
4.3.3	<i>Client response</i>	18
4.4	Router Processing Utilities	19
4.5	Client Response Interpretation	21
4.6	Connection routing and denial	22
	Figure 5: Routing flow chart	22
4.7	ZoneAlarm Pro	23
4.8	Performance considerations	23

FOR "CONFIDENTIAL"

Zone Labs

1 Summary

This proposal summarizes the integration of ZoneAlarm Pro (ZAP) with a vendor's router product line. This integration will provide a best-of-breed security solution to the small business and home networking market. Buyers of router vendor products will now have a more robust security solution not found in competitive hardware - even at two to three times the cost. Implementing and co-marketing this router enhancement will give the router vendor incremental revenue opportunities as well as enhanced customer satisfaction.

The first release of this integration effort will allow small businesses and home networks to ensure each PC is safe from Trojan Horse and Spyware style attacks. By refusing Internet or WAN access to any workstation or server not running ZAP (see Figure 1) security is greatly increased. Optionally, a specific version of ZAP can be required. Exceptions can easily be made for individual devices at the Administrators discretion. When an Internet request is rejected, the user can be routed to a sandbox server where they're given the option to purchase the ZAP or upgrade to the proper version.

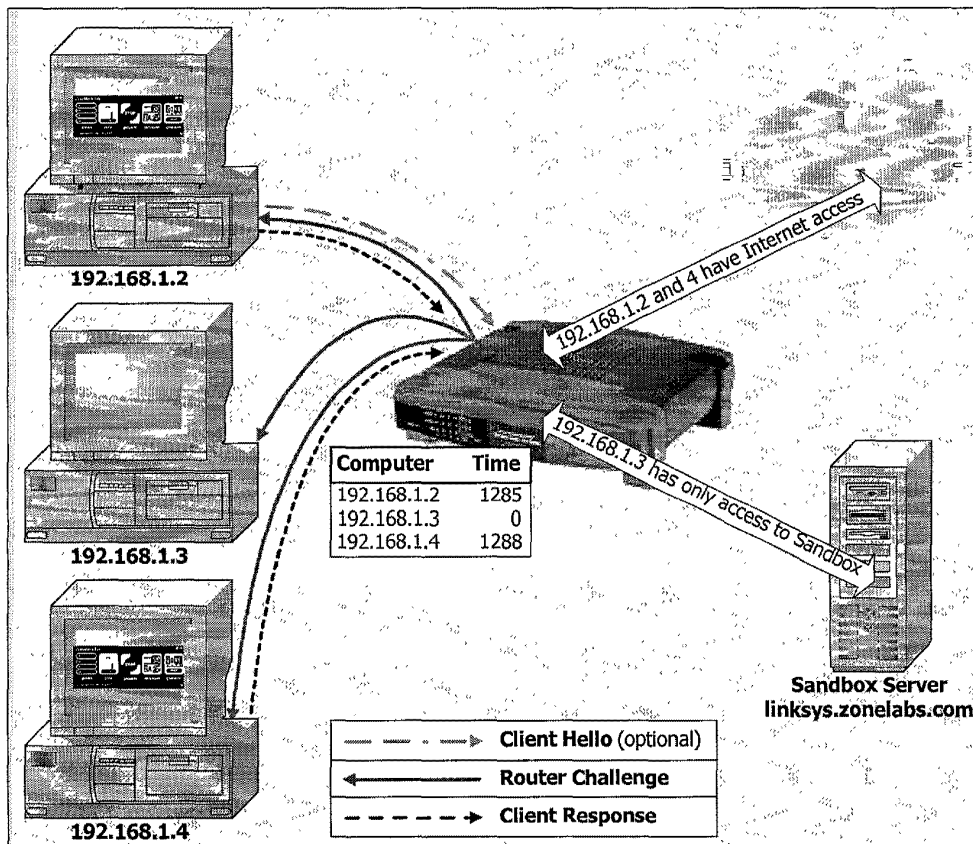


Figure 1: ZAP enforcement protocol

Implementation of this new level of security would simply require a one-time configuration step on the router vendor set-up panels. The single panel set-up screen is intuitive and the

Zone Labs

information needed will be familiar to anyone responsible for configuring the router. Yet the results are a level of security not previously available to small business owners.

Future releases of this integration effort could include options such as; virus detection, logging and log analysis, and centralized administration of policies. This would allow even greater functionality, while still maintaining an attractive price point when compared to current "hardware only" options, SonicWall, for example. A solution such as this would not only help to increase router vendor market share, but would also provide a recurring revenue stream from the sale of ZAP software.

1.1 Key Characteristics:

This combination of technologies provides a more robust security solution than either one alone by offering certain key features:

- *Ease Of Administration* – The skills needed to implement are no greater than those needed to configure the router. *No* additional technical skills or special security expertise is required.
- *Dramatically Improved Security* – By blocking internet access to those machines that are not protected by ZAP, protection against Trojan Horse and Spyware style attacks is greatly increased.
- *Negligible Resource Use* – The "Client Monitoring Protocol" methodology defined within this proposal requires very minimal network traffic.
- *Small Footprint* – The small footprint of the Zone Labs code requires very little of the router's valuable memory space.
- *Scalability* – each PC governs its own security and the router ensures the system is secure.

1.2 Compelling Business factors:

There are several factors that make this an attractive proposal for both the router vendor and Zone Labs.

- *Complementary Technologies* – The complementary technologies of Zone Labs and the router vendor combined, would provide a complete, robust, and easy to manage security solution for the small business.
- *Recurring Revenues* - The router vendor could realize significant recurring revenues from the sale of ZAP to its customers, both existing and new.
- *Increased Market Share* – By positioning this as more complete, more scalable, and more competitive security solution over hardware-only products, the vendor's routers could obtain greater market share.
- *Award Winning Solution* – The opportunity to promote two award-winning products, in combination, could easily be perceived as a positive step by market analysts.

CONFIDENTIAL - INTERNAL

Zone Labs

2 Functional Overview

Without the protection of ZAP, a Trojan Horse or Spyware attack could easily open a PC up to external access. By ensuring the PC has ZAP installed before allowing it access to the Internet or WAN, the risk of this occurring is greatly decreased. In the most simple sense, the proposed integration blocks a PC's access if ZAP is not installed, period. Installation and configuration can be as simple as that. However, some flexibility may be necessary, while maintaining usability without requiring special skills. The flexibility is provided by use of the html configuration panel located in the vendor's Router setup utility (see Figure 2).

Since we are denying users access to the Internet, their browsers would normally display an error code. Instead, we will redirect them to the appropriate URL asking them to either install ZAP or upgrade their current version. A *sandbox server* will provide this functionality, along with additional features.

2.1 Core Features – Internet Access Enforcement:

When the hybrid product is implemented, the router will effectively block all client requests for internet access. Once the client has been validated, then the request will pass. So, like ZAP alone, the default is to block unless identified as safe, as opposed to allowing a free flow of traffic unless identified as a threat.

The process of identifying a "safe" client needs to be both easily configured and flexible enough for the small business owner and home office user. It should be immediately usable and should not require any special skills to configure or maintain. The features afforded by this product would include:

- *Optional Antivirus enforcement –*
 - Ensure that Trend Micro's PC-Cillin is installed
 - Ensure that PC-Cillin's "auto-update" feature is enabled.
 - Ensure that Real-Time monitor is currently running.
 - Set the minimum version of PC-Cillin to be installed.
- *Optional ZAP Version Enforcement –*
 - The option to enforce can be turned on or off at any time. The presence of this product would not require the users to have ZAP installed.
 - The administrator can also require a specific version of ZAP.
- *Device Exemptions –*
 - The ability to enter IP addresses that are exempt from filtering.
- *Simple Configuration –*
 - A single ZoneAlarm Pro Tab on the router's management console to configure the ZAP settings (see figure 2).
 - The options fields are simple and familiar to most users.
- *Installation –*
 - A centralized "one-click" installation method of ZoneAlarm Pro to client machines from a *sandbox server*.
 - "One-click" installation method for virus protection from the *sandbox server*.

Zone Labs

- *Client Monitoring Protocol*–
A “low cost” monitoring methodology will continuously monitor ZAP clients for all necessary information.
- *Centralized License Management*–
 - Centralized licensing and registration for all ZAP clients can be controlled from the router’s management console.
 - Automatic concurrent license management.
- *Retroactively Available*–
 - Current router owners can take advantage of this feature via a firmware update.

FOR CONFIDENTIAL

Zone Labs

NOTE: This screen is only accessible by the router administrator...with a password.

ZoneAlarm Pro SETUP

This screen allows you to configure the router so that the use of ZoneAlarm Pro is enforced for all or selected users. When this option is enabled, the router will prompt the user to install ZoneAlarm Pro before accessing the Internet. For more information see <http://linksys.zonelabs.com/help.htm>.

Enforce PC Security: Enable Disable

Check frequently (more secure) Check less frequently (less bandwidth)

ZoneAlarm Settings: User Prompt:

ZAP Minimum Version Req.: (optional)

License Key: (optional)

Antivirus Settings: PC-Cillin minimum version required:

Enforce PC-Cillin auto-update

Enforce PC-Cillin Real Time Monitor

Exempt Computers: Enable Disable

From:

To:

Figure 2: Router Setup screen

Zone Labs

2.2 ZAP enforcement Overview

When the router with the Zone Labs firmware is installed, ZAP enforcement will be turned off, by default. The administrator will have the option of using the Zone Labs management panel to enable ZAP enforcement. Once ZAP enforcement has been activated, an identification conversation between the router and the client will ensue. If the router detects that the client does not have ZAP installed, the client will be routed to a web page where they will be given the option of downloading ZAP software. The router will continue to re-verify each client using a Client Monitoring Protocol (CMP) conversation. The frequency of this conversation can be set by the administrator.

2.3 Optional ZAP Version Control Overview

The administrator will be able to enforce which version of ZAP is installed on the client machine. If a version older than the one entered is encountered, the user will be re-routed to a *sandbox server* page, which will give them the option to upgrade.

2.4 Optional Licensing Key Management Overview

Administrators will have the ability to enter a license key on the Zone Labs management panel. The key will be encoded for the type of router and the maximum number of concurrent users. If the key is entered ZAP client directly, it will be interpreted as an invalid key.

The key will be sent from the router to the client. The ZAP client will save the key in it's own license key table. Each ZAP client can store multiple keys. ZAP has the ability to search its list of keys and use the "best" one it finds. "Best" being defined as the one that expires latest and offers the most flexibility. If the ZAP client has a non-router restrictive license key installed, it will accept this router key in it's list, but won't use it (won't count as a concurrent user) until the non-router restrictive key expires.

While a) ZAP is running, b) the machine is behind the ZAP enforced enabled router, and c) ZAP's "best" key is the one from the router, it will count as a concurrent user. When any of the above conditions are not met, it does not count.

Error! No topic specified.

Figure 3: determine current user flowchart

If the ZAP client machine is removed from the original LAN, ZAP will detect the absence of the router. The key will still function, outside of the concurrent user count, for 30-days. This is to allow a user on a business trip or working from home to have Internet access using a "licensed" version of ZAP. Once the client is returned to the network, and a ZAP configured router is detected, the client participates as described and the 30-day counter is no longer in effect.

All license key management and concurrent user management will be performed in the ZAP client, not on the router.

Zone Labs

2.5 Optional Virus Checking Overview

The Zone Labs administration panel will contain fields for enforcing virus protection on the client. The available fields will allow enforcement that either PC-Cillin Real-Time Monitor is running and/or PC-Cillin is configured for Auto-Update. The administrator will also be able to enter the minimum version of PC-Cillin that will meet the validation requirements.

2.6 Sandbox Server Overview

Zone Labs will host a web server or *sandbox server* handle client exceptions and software distribution. The purpose of the sandbox server is to:

- Inform the user when Internet access is blocked because ZAP isn't installed on the PC.
- Inform the user when their version of ZAP is outdated.
- Inform the user when they require PC-Cillin to be installed.
- Inform the user when their version of PC-Cillin is outdated.
- Inform the user that they need to enable PC-Cillin Real-Time Monitoring.
- Inform the user that they need to enable PC-Cillin Auto-Update.
- Provide help for installation and administration.
- Allow user download of trial versions of ZAP/PC-Cillin.
- Allow user download of ZAP/PC-Cillin updates.
- Link to e-commerce site Allow users to purchase additional blocks of licenses to increase their number of concurrent users.

The location of the sandbox server will be always be at a fixed location, e.g., <http://router.zonelabs.com>. That server listens on multiple ports and the user is shown an error/information page, depending on which port they are routed to. OI: Either the router will allow DNS access only when the full web-site address matches the sandbox location to allow for re-routing; or the router will forced port routing.

2.7 Future feature enhancements

As the product matures enhancements can include:

- Centralized logging and reporting
- Centralized security policy creation, deployment and enforcement
- Router hosted Zone Labs Firewall functionality with Web-based administrator.
- Remote security administration by service providers using the vendor's routers.
- Internet Access and Network management applications for small businesses.

FOUO "E53E0E03

Zone Labs

3 Procedural Overview

The implementation, configuration, and operation of the ZAP/router integration product are designed to be as simple and transparent as possible.

3.1 Implementation

The implementation, from the customer's point of view, for the ZAP-router integration will follow a simple process.

- 1) For existing router customers, the administrator downloads and installs the router firmware update containing a Zone Labs tab in the Management Console and the CMP routines.
- 2) On the ZAP administration tab, the administrator configures the ZAP management settings.
- 3) Each user installs ZAP on their client machines from the sandbox server.

Once the configuration is complete and the client ZAP is installed client monitoring begins.

- 1) A verification conversation (initiated by either the router or the client, depending on conditions and timing) will ensue between the client and the router.
- 2) The Client Monitor (router routines) informs the client of which options, versions, etc. to enforce.
- 3) The client responds with as required.
- 4) The Client Monitor interprets the response authorizes or denies access with a response (see Figure 3).
- 5) The Client Monitor stores the state of machine for a given period of time (or number of cycles).

FOR INTERNAL USE ONLY

Zone Labs

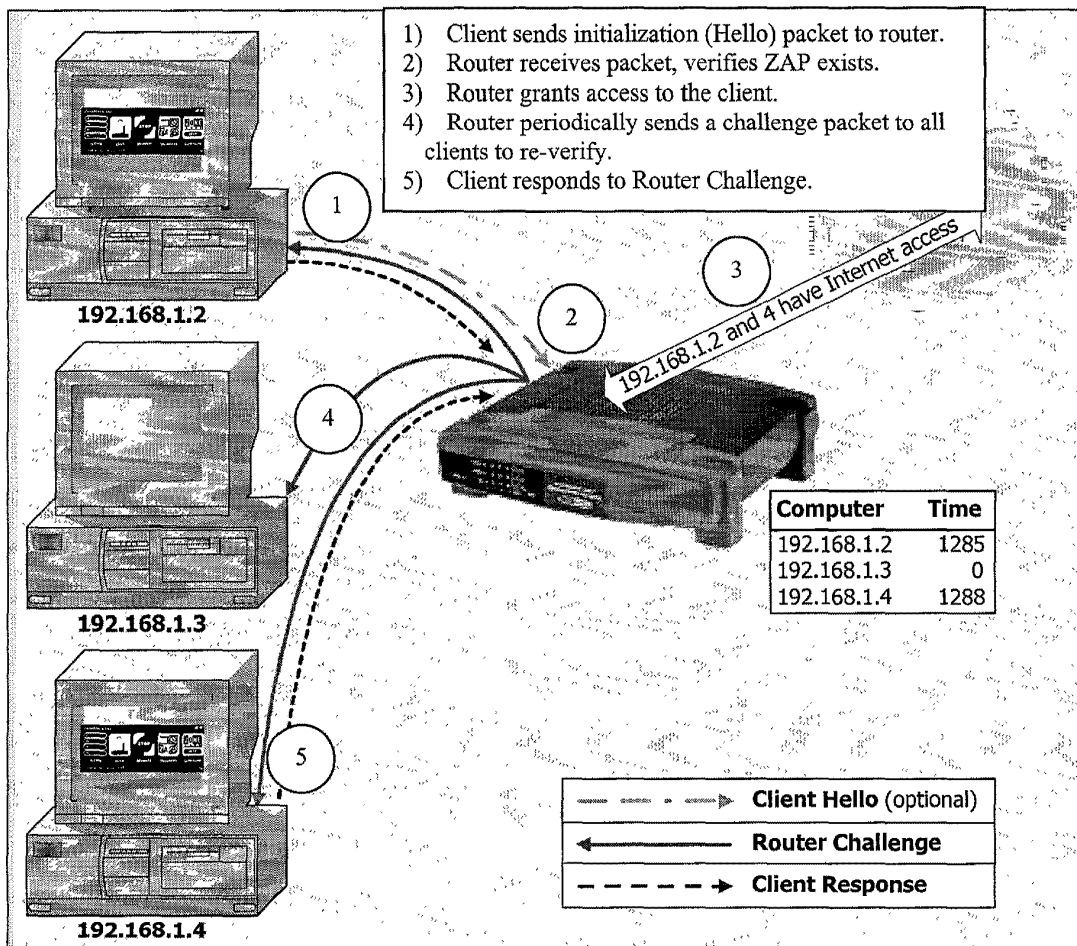


Figure 4: ZAP enforcement protocol

Further details of this implementation can be found in Appendix A: Technical Details.

3.2 Client Monitoring Protocol Overview

In order to enforce the use of ZAP on the LAN side, the router will broadcast an encrypted *router challenge* packet every n number of seconds. A ZAP enabled client will then respond with its own encrypted *client response* packet within $n/2$ seconds. The client response packet will inform the router that it is ZAP enabled and whether or not it meets the minimum-security requirements. The exact response time is randomized to minimize packet collisions on the Ethernet.

When a client machine is placed behind the Zone Labs configured router for the first time, it will wait for a router challenge. When the challenge is received, it will store information about the router for future identification and it will formulate the appropriate client response packet and send it to the router. The longest the client will have to wait for Internet access the very first time it is behind this router is equal to the router challenge frequency plus negligible processing time. Since the router frequency is set in units of seconds, the total time is also negligible.

The next time the machine is booted, ZAP will determine if it is behind the same router (or any ZAP configured router is has communicated with before). If so, it will immediately send a *client*

Zone Labs

hello packet. The router will respond with a router challenge. This will allow there to be no latency in the client's first Internet access request.

Once the router receives the *client response* packet it will timestamp the response in units of seconds since boot of the router. This will be stored in an array of IP addresses in the LAN address space (to be determined by the router coders). The router Client Monitor will analyze the contents of the packet and return an appropriate value based on whether or not the minimum-security configuration is met. This is also stored in the array.

When the NAT portion of the router detects a client attempt to create a new TCP or UDP connection it will first verify determine whether or not the client is on the "exempt list". If it is not, it will be passed the state code sent in the machine's last *client response* packet.

T0900"659609

Zone Labs

4 Technical details

The implementation of the ZAP-router integration will consist of a number of components as described below:

4.1 ZAP enforcement Router administration panel

The suggested user interface for the router is an additional panel in the main screen of the router HTML configuration utility.

Elements of the panel are:

Element	Default	Comment
ZAP enforcement <i>Enable/Disable</i>	Disable	When disabled, the router's operations are unaffected. Once activated each user will automatically be prompted to download/install a trial version of ZAP when they try to access the Internet via HTTP.
Monitoring frequency	Check Frequently.	Determines how often the router will broadcast a <i>router challenge</i> packet. There will be two choices for the user: "Check frequently (more secure)" "Check less frequently (less bandwidth)"
User prompt <i>Text string</i>	TBD	Determines a prompt in ZAP to inform the user that a router enforces the security or any other text an administrator might choose.
ZAP version <i>Version string</i>	Empty	Allows an administrator to determine the minimum acceptable version of ZAP.
ZAP license key	Empty	Optional entry to allow automatic distribution of the ZAP license key. Keys will be encoded with the maximum number of concurrent users.
ZAP Buy Button	Button	Link to an e-commerce site to purchase a new ZAP license pack.
PC-Cillin Version	Empty	Allows the administrator to determine the minimum acceptable version of PC-Cillin.
PC-Cillin Real-Time Monitor Enforcement	Disable	A checkbox to enforce that PC-Cillin Real-Time Monitor is enforced.
PC-Cillin Auto-Update Enforcement	Disable	A checkbox to enforce that PC-Cillin Real-Time Monitor is enforced.
Exempted devices <i>IP addresses range</i>	Empty	The range of IP addresses for devices that is exempt from ZAP enforcement. This allows the use of none-Windows PCs and other devices in this environment. The exempt devices should be in the LAN address range and not be assigned by DHCP (unless DHCP allows reserved addressed).

Table 1: User Interface Elements

Zone Labs

4.2 Sandbox server

The sandbox server is implemented as a server on the Internet ("router.zonelabs.com"). Zone Labs will build and host this server.

ZL Deliverable: ZL will host, maintain and update the Sandbox Server.

Future versions of the sandbox server will also be able to respond to protocols other than HTTP in order to further improve the user experience. For example, a user that has his/her security disabled and tries to access the Internet via RealNetwork's protocol could get an appropriate voice prompt.

The server listens on port 80 and 8080 to 8112 and responds with different HTTP pages to the various ports:

Port	Content
80	General help and trouble shooting
8080	FAQs
8081	Reserved
8082	Prompts user to install ZAP
8083	Prompts user to update ZAP with new version
8084	Informs user their license key is invalid – contact administrator
8085	Informs user that the ZAP license is insufficient for number of users. Asks them to contact their administrator.
8086	Informs user that he/she needs to download anti-virus software
8087	Informs user to update antivirus software
8088	Informs user to activate Real-Time monitoring
8089	Informs user to activate antivirus Auto-Update
8090	For future use
8091	For future use
8092-8112	Prompts user to troubleshoot installation etc.

Table 2: LAN connection logic

FOR "CONFIDENTIAL"

Zone Labs

4.3 Client monitoring protocol (CMP)

The proposal includes a simple monitoring protocol to:

- Assure that ZAP is installed on the appropriate client PCs.
- Assure that ZAP has the correct version.
- Communicate any settings to the clients.

The CMP is roughly modeled after BOOTP or DHCP. It will use UDP protocol, port 491 on both router and client. Every packet is encrypted using the router's key and decrypted using the client's key or visa versa.

Each packet consists of a header, body and optional additional parameters. This ensures expandability and interoperability even if the router and clients use different versions of the protocol. Options have the following format:

Element	Size	Comment
Option ID	WORD	Specific to option
Option size	WORD	Size of this structure including data
Option data		Specific to option

Table 3: Options

As mentioned before, the protocol uses port 491, UDP on both router and clients.

ZL Deliverable: Zone Labs will furnish all routines needed to implement this protocol.

4.3.1 Client Hello

This packet is sent by the client to the router to request a challenge.

If a ZAP client has previously been managed by a router, the client will remember the IP and MAC address of that router. The next time the client starts up and encounters the same subnet; it will proactively let the router know that it needs a challenge using a Client Hello packet. This reduces the number of heartbeats sent by the router and reduces the access time to the Internet at client startup.

Element	Size	Comment
Packet ID	WORD	CLIENT_HELLO (== 1)
Packet size	WORD	Size of this structure + options
Protocol version	WORD	PROTOCOL_VERSION (== 1)
Packet options	WORD	Number of options following this structure
Packet CRC	DWORD	Checksum for the packet
Sender IP address	DWORD	In network byte order - prevents spoofing
Sender Product ID	DWORD	Assigned by ZL (= 0x80000001 for ZAP)
Sender version	DWORD +WORD	Actual ZAP version (Union of DWORD + WORD)
Reserved	DWORD	Reserved
Options		

Table 4: Client hello packet

Zone Labs

4.3.2 Router challenge

The router sends this packet to either an individual client or to all clients that expect a response to permit Internet access.

When ZAP enforcement is enabled, the browser will broadcast a packet on the LAN side to the local broadcast address (IP address || (^IP Mask)), UDP port 491 every N seconds, as determined by the Monitoring Frequency setting. The first broadcast packet will be sent as soon as possible after the router's boot.

The packet will also be sent to an individual client as a response to a *client hello* packet.

ZL Deliverable: The packet will be encrypted by the router's public key. The code to prepare and encrypt the packet will be furnished by Zone Labs. It will be decrypted by the ZAP client using the client's private key. The packet has the following structure:

Element	Size	Comment
Packet ID	WORD	ROUTER_CHALLENGE (== 2)
Packet size	WORD	Size of this structure + options
Protocol version	WORD	PROTOCOL_VERSION (== 1)
Packet options	WORD	Number of options following this structure
Packet CRC	DWORD	Checksum for the packet
Sender IP address	DWORD	In network byte order - prevents spoofing
Sender Product ID	DWORD	Assigned by ZL (= 0x00000001 for router)
Sender version	DWORD	Vendor router version
Router Session ID	DWORD	Per router session generated random value
Response time	DWORD	Time in seconds in which the router expects a response.
Timestamp	DWORD	Packet Timestamp
Reserved	DWORD	
Options		

Table 5: Router challenge packet

The "client version" allows the administrator or the router to request a specific minimum acceptable client version. Multiple records of the option are acceptable:

Element	Size	Comment
Option ID	WORD	OPTION_CLIENT_VERSION (== 1)
Option size	WORD	16
Product ID	DWORD	Assigned by ZL, 0x80000001 for ZAP
Product version	4xWORD	Minimum acceptable product version

Table 6: Client Version Option

Zone Labs

The "license option" allows the administrator to use the router for automatic distribution of a license key. ZAP automatically installs that key:

Element	Size	Comment
Option ID	WORD	OPTION_ZL_LICENSE (== 2)
Option size	WORD	34
ZAP License key	CHAR[28]	Set by admin, concatenated to a single string
Number of users	WORD	Contains the number of ZAP licenses, that are currently issued by the router

Table 7: ZL License Option

The "user prompt" allows the administrator to use the router for automatic distribution of a user prompt that will be displayed in the ZAP UI.

Element	Size	Comment
Option ID	WORD	OPTION_USER_PROMPT (== 3)
Option size	WORD	64
User prompt	CHAR[60]	String to display to user in ZAP

Table 8: User prompt option

The "anti virus challenge" option allows the administrator to use the router for anti-virus enforcement and distribution. ZoneAlarm Pro has the appropriate code to verify if the AV program is running and both program file and data file are up to date.

Element	Size	Comment
Option ID	WORD	ANTI_VIRUS_CHALLENGE (== 4)
Option size	WORD	32
Antivirus Product ID	DWORD	Code to identify AntiVirus name (TRENDMICRO == 1)
Antivirus version	4xWORD	Optional: CMPVERS "7.0.0"
Antivirus Auto-Update Enforcement	BOOL	Optional: Boolean to identify if the Auto-Update enforcement option is selected
Antivirus Real-Time Monitoring Enforcement	BOOL	Optional: Boolean to identify if the Real-Time Monitor enforcement option is selected
Antivirus Reserved1	DWORD	Reserved
Antivirus Reserved2	DWORD	Reserved

Table 9: Anti virus challenge option

Trend Micro

Zone Labs

4.3.3 Client response

This packet is sent by the client to the router as a response to the router challenge.

ZL Deliverable: The router will then decrypt, verify and interpret each client response and store the result in its client array. Zone Labs will furnish the code for this. The packet has the following structure:

Element	Size	Comment
Packet ID	WORD	CLIENT_RESPONSE (== 3)
Packet size	WORD	Size of this structure + options
Protocol version	WORD	PROTOCOL_VERSION (== 1)
Packet options	WORD	Number of options following this structure
Packet CRC	DWORD	Checksum for the packet
Sender IP address	DWORD	In network byte order - prevents spoofing
Sender Product ID	DWORD	Assigned by ZL (= 0x80000001 for ZAP)
Sender version	DWORD	ZAP version
Router Session ID	DWORD	Copied from router challenge
Challenge Timestamp	DWORD	Timestamp copied from router challenge packet.
Status	DWORD	Client Status (see <i>Client Response Interpretation</i> , section 4.5 below)
Reserved	DWORD	Reserved (==0)
Options		

Table 10: Router challenge packet

CONFIDENTIAL

Zone Labs

4.4 Router Processing Utilities

The router will need to perform several short or “small” functions. The include:

- Encrypting packets
- Decrypting packets
- Building *Router Challenges*
- Analyzing decrypted packets
- Making the Pass/Re-route decision

The Encrypt Packet function (CMPEncryptPacket) will be defined as follows:

Element	Size	Comment
Input buffer	UCHAR *	The buffer that contains the packet to be encrypted.
Output buffer	UCHAR *	The buffer that contains the encrypted packet.
Buffer length	DWORD	The length of the of the output buffer

Table 11: CMPEncryptPacket function definition

The Decrypt Packet function (CMPDecryptPacket) will be defined as follows:

Element	Size	Comment
Input buffer	UCHAR *	The buffer that contains the packet to be decrypted.
Output buffer	UCHAR *	The buffer that contains the decrypted packet.
Buffer length	DWORD	The length of the of the output buffer

Table 12: CMPDecryptPacket function definition

The Create Router Challenge Packet function (CMPCreateChallengePacket) will be defined as follows:

Element	Size	Comment
Input buffer	UCHAR *	The buffer that contains the packet to be decrypted.
Output buffer	UCHAR *	The buffer that contains the decrypted packet.
Buffer length	DWORD	The length of the of the output buffer

Table 13: CMPCreateChallengePacket function definition

Part of this processing will include determining the number of licenses currently held by counting them in the state table and passing that count to ZAP (see **Router challenge**, section 4.3.2 above for packet structure).

FOR "SECRET"

Zone Labs

The Analyze decrypted packet function (CMPAnalyseDecryptedPacket) will be defined as follows:

Element	Size	Comment
Input buffer	UCHAR *	The buffer that contains the decrypted packet to be analyzed.
Client Table buffer	UCHAR *	The buffer that contains the client state table.

Table 15: CMPAnalyseDecryptedPacket function definition

It will determine the package is a:

- *Client Response* packets (see *Client response*, section 4.3.3 above) or a
- *Client Hello* packets (see *Client Hello*, section 4.3.1 above)

and will process accordingly (see *Client Response Interpretation*, section 4.5 below).

The function that will determine whether or not to pass an Internet request (CMPAuthorizeTraffic) will be defined as follows:

Element	Size	Comment
IP Address	DWORD	The IP Address requesting Internet access.
Client Table Buffer	UCHAR *	The buffer that contains the client state table.

Table 16: CMPAuthorizeTraffic function definition

TABLE "TABLE"

Zone Labs

4.5 Client Response Interpretation

The routine that verifies and interprets the client response packet returns a DWORD. Unless the result is "Packet invalid" the result should be stored in the client list. The value should be interpreted as follows:

Response	Value	Comment
Packet invalid	1	Packet corrupted, spoofed etc., ignore result
No ZAP is installed	2	Sandbox: Download and install ZAP
ZAP version outdated	3	Sandbox: Download newer version
Invalid license	4	Sandbox: Invalid license key, contact administrator.
License exceeded	5	Sandbox: Buy more blocks of ZAP user licenses.
No Antivirus program installed	6	Sandbox: Download and install PC-Cillin (this option may not be used).
Antivirus wrong version	7	AV wrong version
Antivirus Real-Time Monitor not running	8	AV Real-Time Monitor not running on client
Antivirus Auto-Update not configured	9	AV Auto-Update is not configured.
Packet time stamp	> 256	Current time stamp in seconds + 256
Client exempt	-1	Used internally to mark exempt clients

Table 17: Client response results

The router will store the result in a client table. This table could be an extension of the existing NAT or ARP table or independent from those.

Every time a client PC attempts a connection with the Internet, the router will look up the result of the client response and verify that the client has Internet access before it applies the NAT-related IP header changes etc.

TOP SECRET

Zone Labs

4.6 Connection routing and denial

If the client has no Internet access, then the router will reroute the client to a sandbox before the NAT-related IP header changes or denies the connection altogether:

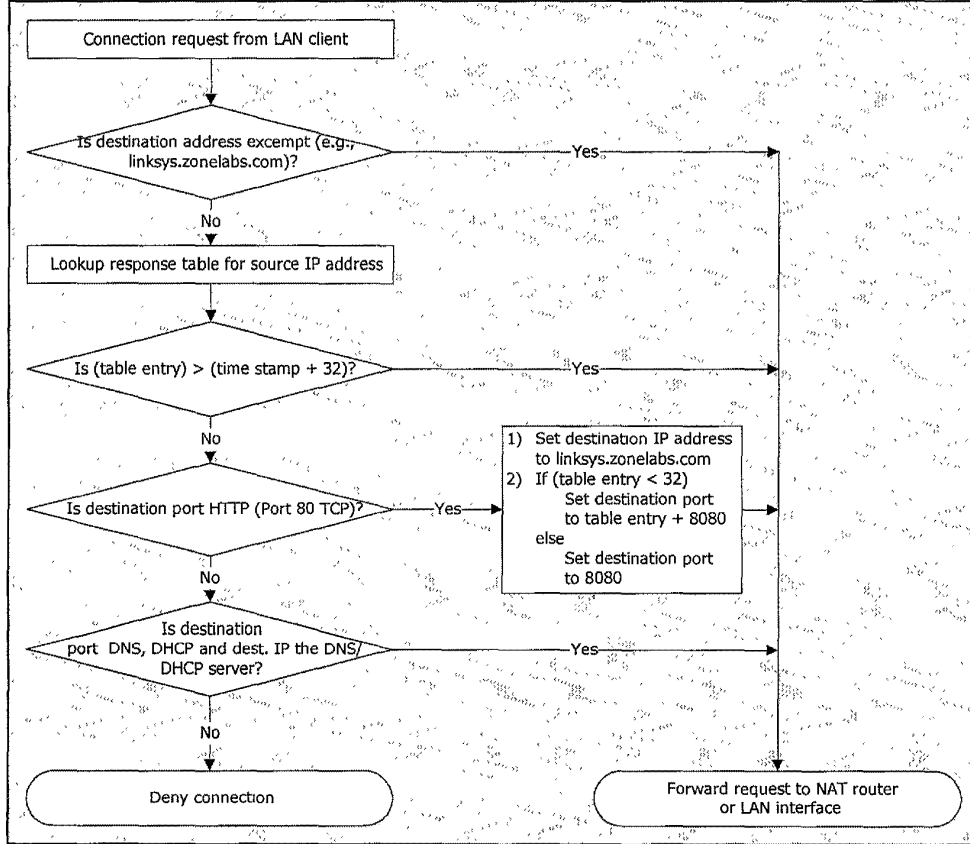


Figure 5: Routing flow chart

ZL provides the code needed for the implementation of this logic.

CONFIDENTIAL

Zone Labs

4.7 ZoneAlarm Pro

ZAP will listen to the router challenge broadcast. This part will probably be implemented in the firewall so we can filter by the MAC address of the router. The FW will then forward the decrypted packet as an FW message to TrueVector (TV). TV then verifies the rest of the contents and its own settings and then sends back a response packet at a random time based on the heartbeat frequency.

4.8 Performance considerations

The bandwidth impact of the proposed enforcement mechanism on a 10BaseT network should be less than 2% of the total bandwidth. On a 100BaseT network the impact should be less than 0.2%.

FOR "CONFIDENTIAL"

```

/*****
/**          Zone Labs Monitor          **/
/**          Copyright(c) Zone Labs, 2001          **/
*****/

/*
   vsadapter.cpp (vsmon.exe)

       Handles adapter functionality

*/

#include "vsmonpch.h"
#pragma hdrstop
// #include <windows.h>
// #include <windowsx.h>
// #include "vsutil.h"
// #include "vsmon.h"
// #include "vsinet.h"
// #include "vsutil.h"
// #include "vsmon.h"
#include "firewall.h"
#include "vsruleapi.h"
#include "version.rc"

extern void *hLocalContext;

// from vsras.cpp
BOOL WINAPI InitRas(DWORD dwFlags);
VOID WINAPI ExitRas(VOID);
BOOL WINAPI GetRASConnectionName(DWORD dwIPAddr, CHAR *pcName, DWORD dwLen);

// gateway message stuff
INT igWSPProto = 0; // protocol
SOCKET sockGW = INVALID_SOCKET; // socket
USHORT usGWSeq = 0; // sequence number

// cmp stuff
BOOL bCMPEnabled = FALSE; // CMP enabled
DWORD dwCMPCount = 0; // CMP monitored network count

#ifdef _DEBUG

// if _NETWORK_DEBUG is defined, we display debugging info when networks are
// updated
// #define _NETWORK_DEBUG

// if _CMP_DEBUG is defined, we can pretend that a specified gateway is a
// supported router
#define _CMP_DEBUG
#ifdef _CMP_DEBUG
DWORD dwCMPDebugType = 0;
DWORD dwCMPDebugGateway = 0;
#endif
#endif // _DEBUG

```

```

BOOL InitAdapter(DWORD dwFlags)
{
    BOOL bRes = TRUE;
    InitWS(0);
    if (sockGW == INVALID_SOCKET)
    {
        sockGW = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
        if (sockGW != INVALID_SOCKET)
            iGWSPROTO = IPPROTO_ICMP;
        else
        {
            sockGW = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
            if (sockGW != INVALID_SOCKET)
                iGWSPROTO = IPPROTO_UDP;
            else
                bRes = FALSE;
        }
    }
    if (bRes)
    {
        INT iTimeout = 1000;
        setsockopt(sockGW, SOL_SOCKET, SO_SNDTIMEO, (char *)
&iTimeout, sizeof(iTimeout));
    }
    InitRas(0);
    DWORD dwTVFlags = 0;
    RulesGetPropDWORDEx((HCONTEXT) hLocalContext, 0,
PROPID_TV_FUNCTIONALITY, &dwTVFlags);
    bCMPEnabled = (dwTVFlags & TVF_CMP) != 0;
#ifdef _CMP_DEBUG
    // check to see if we should pretend there's a supported router
    dwCMPDebugType = GetRegistryValueDWord(HKEY_LOCAL_MACHINE,
HKEY_VIVA_TRUEVECTOR, "CMPDebugType", 0);
    if (dwCMPDebugType)
    {
        // get the router's ip address
        CHAR cCMPDebugGateway[16];
        if (GetRegistryValueString(HKEY_LOCAL_MACHINE, HKEY_VIVA_TRUEVECTOR,
"CMPDebugGateway",
cCMPDebugGateway, sizeof(cCMPDebugGateway)) &&
cCMPDebugGateway[0])
        {
            dwCMPDebugGateway = StrToIPAddr(cCMPDebugGateway);
            if (dwCMPDebugGateway)
                bCMPEnabled = TRUE;
        }
    }
#endif // _CMP_DEBUG
    return bRes;
}

VOID ExitAdapter(VOID)
{
    dwCMPCount = 0;
    bCMPEnabled = FALSE;
    ExitRas();
}

```

T03070 "S9E0E0E0"

```

    iGWSPROTO = 0;
    if (sockGW != INVALID_SOCKET)
    {
        closesocket(sockGW);
        sockGW = INVALID_SOCKET;
    }
    ExitWS();
}

```

```

BOOL WINAPI SendDiscard(DWORD dwIPAddr, SOCKET sock, USHORT usId, USHORT
usSeq)
//
// Send discard packet to dwIPAddr
//
{
    BOOL bClose = FALSE;
    if (sock == INVALID_SOCKET)
    {
        if ((sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) ==
INVALID_SOCKET)
            return FALSE;

        bClose = TRUE;
    }

    struct sockaddr_in DstAddr;
    memset(&DstAddr, 0, sizeof(DstAddr));
    DstAddr.sin_family = AF_INET;
    DstAddr.sin_addr.s_addr = dwIPAddr;
    DstAddr.sin_port = htons(IPPORT_DISCARD);

    INT iData[2] = {usId, usSeq};
    BOOL bRes = (sendto(sock, (const char *) iData, sizeof(iData), 0, (const
struct sockaddr *) &DstAddr, sizeof(DstAddr)) != SOCKET_ERROR);

    if (bClose)
        closesocket(sock);

    return bRes;
}

```

```

BOOL WINAPI ArpGateway(PIP_INFO pIpInfo)
{
    if (pIpInfo->dwGateway && IS_PRIVIP(pIpInfo->dwGateway) &&
(pIpInfo->dwGateway != pIpInfo->dwAddr))
        return TRUE;
    return FALSE;
}

```

```

VOID WINAPI PingGateway(DWORD dwGWAddr)
{
    switch (iGWSPROTO)
    {
        case IPPROTO_UDP:

```

```

        FirewallAddStateFromInfo(NET_PROTO_IP_UDP, dwGWAddr, 0,
htons(IPPORT_DISCARD), 0, 0, FW_TTL_TEMP);
        SendDiscard(dwGWAddr, sockGW, (USHORT)
GetCurrentProcessId(), ++usGWSeq);
        break;
    case IPPROTO_ICMP:
        FirewallAddICMPStateFromInfo(dwGWAddr, 0, FW_ICMP_ECHO);
        SendPing(dwGWAddr, sockGW, (USHORT) GetCurrentProcessId(),
++usGWSeq);
        break;
    default:
        break;
    }
}

```

```

PCMP_CLIENT WINAPI AddCMPClient(PIP_INFO pIpInfo, PVR_NETWORK pNetwork)
{
    if (!pIpInfo || !pNetwork || !bCMPEnabled || pNetwork->pCMPClient ||
!ArpGateway(pIpInfo))
        return NULL;

    DWORD dwGWType = EthAddrToGWType(pNetwork->GWPhysAddr.Eth);
#ifdef _CMP_DEBUG
    // check to see if we should pretend this is a supported router
    if (!dwGWType && dwCMPDebugType && (dwCMPDebugGateway ==
pIpInfo->dwGateway))
        dwGWType = dwCMPDebugType;
#endif
    if (!dwGWType)
        return NULL;

    LPVOID lpKey = GetCMPEncryptionKey(dwGWType);
    if (!lpKey)
        return NULL;

    DWORD dwMonStatus = pNetwork->GetDWordValue(GDV_MONSTATUS);
    if (dwMonStatus == NWMS_EXEMPT)
        return NULL;

    WORD wVers[4] = {VER_PRODUCTVERSION};
    PCMP_CLIENT pClient = new CMP_CLIENT(pNetwork->dwID,
dwMonStatus,
lpKey,
dwGWType,
CMPP_ZAPRO,
(PCMPVERS)

wVers,

pIpInfo->dwGateway,

pIpInfo->dwAddr,

htons(CMP_PORT));
    if (pClient && (pClient->dwStatus == -1))
    {
        delete pClient;
        pClient = NULL;
    }
}

```

```

    }
    if (pClient)
    {
        dwCMPCount++;
        pNetwork->pCMPCClient = pClient;
    }
#ifdef _CMP_DEBUG
    CHAR cMsg[256];
    wsprintf(cMsg, "AddCMPCClient: pNetwork=%x pClient=%x n=%lu\n", pNetwork,
pClient, dwCMPCount);
    OutputDebugString(cMsg);
#endif
    return pClient;
}

```

```

VOID WINAPI SyncCMPClients(BOOL bEnabled)
{
    if (bEnabled == bCMPEnabled)
        return;

    bCMPEnabled = bEnabled;
    if (bCMPEnabled)
    {
        // cmp enabled - create clients
        PVR_ADAPTER pAdapter = FindFirstAdapter();
        while (pAdapter)
        {
            for (DWORD j = 0; j < pAdapter->dwIpCount; j++)
                AddCMPCClient(&(pAdapter->IpInfo[j]),
pAdapter->Networks[j]);
            pAdapter = (PVR_ADAPTER) pAdapter->pNext;
        }
    }
    else if (dwCMPCount)
    {
        // cmp disabled - toss clients
        dwCMPCount = 0;
        PVR_NETWORK pNetwork = FindFirstNetwork();
        while (pNetwork)
        {
            if (pNetwork->pCMPCClient)
            {
                delete pNetwork->pCMPCClient;
                pNetwork->pCMPCClient = NULL;
            }
            pNetwork = (PVR_NETWORK) pNetwork->pNext;
        }
    }
}

```

```

VR_ADAPTER::VR_ADAPTER(
    PVR_ADAPTER *ppAdapter,
    PIP_ADAPTER pIPAdapter,
    DWORD dwFlags)
{
    dwAdapterStatus = 0xffffffff;
}

```

```

RegisterObject(VT_ADAPTER | VT_DONTNOTIFY, (PVR_BASE *) ppAdapter);
IpInfo[0].dwAddr = 0; //make sure
memset(&Networks, 0, sizeof(Networks));
SetAdapterInfo(pIPAdapter, dwFlags);
}

VR_ADAPTER::~VR_ADAPTER()
{
}

VOID VR_ADAPTER::DestroyObject(void)
{
    UnRegisterObject();

    BASE::DestroyObject();
}

BOOL CompareAdapterInfo(PIP_ADAPTER pIPAdapter1, PIP_ADAPTER pIPAdapter2)
{
    if (memcmp(pIPAdapter1->cDesc, pIPAdapter2->cDesc, MAX_IP_DESC_LEN))
        return TRUE;
    if (memcmp(pIPAdapter1->cName, pIPAdapter2->cName, MAX_IP_NAME_LEN))
        return TRUE;
    if (pIPAdapter1->dwIndex != pIPAdapter2->dwIndex)
        return TRUE;
    if (pIPAdapter1->dwFWId != pIPAdapter2->dwFWId)
        return TRUE;
    if (pIPAdapter1->dwIFType != pIPAdapter2->dwIFType)
        return TRUE;
    if (pIPAdapter1->dwIpCount != pIPAdapter2->dwIpCount)
        return TRUE;
    if (memcmp(&(pIPAdapter1->IpInfo), &(pIPAdapter2->IpInfo), MAX_IP_ADDR *
sizeof(IP_INFO)))
        return TRUE;
    if (memcmp(&(pIPAdapter1->PhysAddr), &(pIPAdapter2->PhysAddr),
sizeof(PHYS_ADDR)))
        return TRUE;
    return FALSE;
}

BOOL VR_ADAPTER::SetAdapterInfo(PIP_ADAPTER pIPAdapter, DWORD dwFlags)
{
    BOOL bRes = FALSE;
    dwAdapterFlags = dwFlags & ADAPTER_INFO_MASK;
    if (CompareAdapterInfo(pIPAdapter, this)) // if something changed in the
adapter
    {
        bRes = TRUE;
        if (dwGlobalSSFlags & SSF_LLD_FIREWALL)
        {
            // tell the firewall our IP address(es) have changed
#ifdef _NETWORK_DEBUG
            CHAR cNWMsg[256];
            wsnprintf(cNWMsg, sizeof(cNWMsg), "SetAdapterInfo(n=%lu
id=%lx p=%08lx): ip0=%08lx=>%08lx\n",

```

```

        pIPAdapter->dwIndex, dwID, this, IpInfo[0].dwAddr,
pIPAdapter->IpInfo[0].dwAddr);
        OutputDebugString(cNWMsg);
#endif
        if (IpInfo[0].dwAddr)
            FirewallDelLocalIP(0, this);
        if (pIPAdapter->IpInfo[0].dwAddr)
            FirewallAddLocalIP(pIPAdapter);
        // update arp table for new gateway(s)
        if (!(dwAdapterFlags & ADAPTER_INFO_DIALUP))
        {
            for (DWORD i = 0; i < MAX_IP_ADDR; i++)
            {
                if (IpInfo[i].dwGateway !=
pIPAdapter->IpInfo[i].dwGateway)
                {
#ifdef _NETWORK_DEBUG
                    wsprintf(cNWMsg, sizeof(cNWMsg),
"SetAdapterInfo: gw(%lu)=%08lx=>%08lx\n",
                    i, IpInfo[i].dwGateway,
pIPAdapter->IpInfo[i].dwGateway);
                    OutputDebugString(cNWMsg);
#endif
                    if (ArpGateway(&IpInfo[i]))
                        FirewallArpTableDel(IpInfo[i].dwGateway);
                    if (ArpGateway(&(pIPAdapter->IpInfo[i])))
                    {
                        FirewallArpTableAdd(pIPAdapter->IpInfo[i].dwGateway);
                        PingGateway(pIPAdapter->IpInfo[i].dwGateway);
                    }
                }
            }
        }
        // save the new adapter IP address info
        *((PIP_ADAPTER) this) = *pIPAdapter;
    }
    return bRes;
}

```

```

BOOL VR_ADAPTER::InList(DWORD dwList)
{
    switch (dwList)
    {
        case AL_ALL:
            return TRUE;
        case AL_ACTIVE:
            return IpInfo[0].dwAddr != 0; // subnet mask?
        case AL_LAN:
            return (IpInfo[0].dwAddr != 0) && (dwAdapterStatus != 0);
        // subnet mask?
        default:
            break;
    }
}

```



```

        if ((dwIPAddr & IpInfo[i].dwMask) == (IpInfo[i].dwAddr &
IpInfo[i].dwMask))
            return TRUE;
    }
}
return FALSE;
}

```

```

BOOL VR_ADAPTER::IsSubnetBroadcast(DWORD dwIPAddr)
{
    if (!dwIPAddr) //check for 0xffffffff?
        return FALSE;

    if (!IpInfo[0].dwAddr)
        return FALSE;

    for (DWORD i = 0; i < dwIpCount; i++)
    {
        if ((IpInfo[i].dwMask != 0xffffffff) && (dwIPAddr ==
SUBNET_BROADCAST_ADDR(IpInfo[i].dwAddr, IpInfo[i].dwMask)))
            return TRUE;
    }

    return FALSE;
}

```

```

BOOL VR_ADAPTER::IsNetBroadcast(DWORD dwIPAddr)
{
    if (!dwIPAddr) //check for 0xffffffff?
        return FALSE;

    if (!IpInfo[0].dwAddr)
        return FALSE;

    DWORD dwBCAddr;
    for (DWORD i = 0; i < dwIpCount; i++)
    {
        if (IS_CLASSC(IpInfo[i].dwAddr))
            dwBCAddr = CLASSC_BROADCAST_ADDR(IpInfo[i].dwAddr);
        else if (IS_CLASSB(IpInfo[i].dwAddr))
            dwBCAddr = CLASSB_BROADCAST_ADDR(IpInfo[i].dwAddr);
        else if (IS_CLASSA(IpInfo[i].dwAddr))
            dwBCAddr = CLASSA_BROADCAST_ADDR(IpInfo[i].dwAddr);
        else
            continue;

        if (dwIPAddr == dwBCAddr)
            return TRUE;
    }

    return FALSE;
}

```

```

BOOL VR_ADAPTER::GetAdapterTitle(CHAR *pChar, DWORD dwLen)

```

```

{
    CHAR cAddr[16];
    CHAR cMask[16];
    IPAddrToStr(IpInfo[0].dwAddr, cAddr, sizeof(cAddr));
    IPAddrToStr(IpInfo[0].dwMask, cMask, sizeof(cMask));
#if 1
    CHAR *pcDesc = cDesc;
    DWORD dwIndex = strlen(cDesc);
    CHAR cDesc2[MAX_IP_DESC_LEN] = {0};
    if (dwIndex && (cDesc[dwIndex - 1] == ' '))
    {
        dwIndex--;
        pcDesc = cDesc2;
        lstrcpy(cDesc2, cDesc, sizeof(cDesc2));
        while (cDesc2[dwIndex] == ' ')
        {
            cDesc2[dwIndex] = 0;
            if (!dwIndex)
                break;
            dwIndex--;
        }
    }
    wsnprintf(pChar, dwLen, "%s (%s/%s%s)", pcDesc, cAddr, cMask, (dwIpCount
> 1) ? ", ..." : "");
#else
    wsnprintf(pChar, dwLen, "%s (%s/%s%s)", cDesc, cAddr, cMask, (dwIpCount >
1) ? ", ..." : "");
#endif
    return TRUE;
}

```

```

BOOL VR_ADAPTER::GetCharValue(DWORD dwProperty, CHAR *pChar, DWORD dwLen)
{
    CHAR* pValue = NULL;
    DWORD dwID = 0;

    switch (dwProperty)
    {
        case GCV_NAME:
            pValue = cName;
            break;
        case GCV_TITLE:
            return GetAdapterTitle(pChar, dwLen);
        case GCV_DESCRIPTION:
            pValue = cDesc;
            break;
        case GCV_ADAPTERID:
            return GetAdapterCID(this, pChar, dwLen);
        case GCV_PHYSADDR:
            if (dwLen >= sizeof(PhysAddr))
            {
                memcpy(pChar, &PhysAddr, sizeof(PhysAddr));
                return TRUE;
            }
            break;
        case GCV_IPDATA:
            if (dwLen >= sizeof(IpInfo))

```



```

        return 0;
        for (DWORD i = 0; i < dwIpCount; i++)
            if (!IS_PRIVIP(IpInfo[i].dwAddr))
                return 0;
        return 1;
    }
    case GDV_ADAPTERDRV:
        return dwFWID;
    case GDV_IPCOUNT:
        return dwIpCount;
    default:
        return BASE::GetDWordValue(dwProperty);
}
}

```

```

DWORD VR_ADAPTER::SetDWordValue(DWORD dwProperty, DWORD dwNewValue)
{
    DWORD dwOldValue = 0;
    switch(dwProperty)
    {
        case GDV_STATUSCODE:
            dwOldValue = dwAdapterStatus;
            dwAdapterStatus = dwNewValue;
            break;
        case GDV_DEFSTATUS:
        case GDV_DEFSTATUS_PRIVATE:
            // can't set these
            break;
        default:
            return BASE::SetDWordValue(dwProperty, dwNewValue);
    }
    return dwOldValue;
}

```

```

// returns the physical address of the given IP address gateway
//
BOOL GetGatewayPhysAddr(DWORD dwIPAddr, PHYS_ADDR &physAddr)
{
    DWORD dwBytesReturned = 0;
    ZeroMemory(&physAddr, sizeof(physAddr));
    FirewallGetArpInfo(dwIPAddr, physAddr.Eth, sizeof(physAddr.Eth),
&dwBytesReturned);
#ifdef _NETWORK_DEBUG
    CHAR cGWPAMsg[256];
    wsnprintf(cGWPAMsg, sizeof(cGWPAMsg), "GetGatewayPhysAddr: ip=%08lx
eth=%02x-%02x-%02x-%02x-%02x-%02x\n",
            dwIPAddr,
            physAddr.Eth[0],
            physAddr.Eth[1],
            physAddr.Eth[2],
            physAddr.Eth[3],
            physAddr.Eth[4],
            physAddr.Eth[5]);
    OutputDebugString(cGWPAMsg);
#endif // _NETWORK_DEBUG
    return dwBytesReturned == sizeof(physAddr.Eth);
}

```

"00000000"

```

}

#ifdef _NETWORK_DEBUG

#define NWS_NEW          0
#define NWS_FOUND       1
#define NWS_DELETE      2

VOID WINAPI DumpNetworkInfo(PVR_ADAPTER pAdapter, DWORD dwIndex, PVR_NETWORK
pNetwork, DWORD dwNWS, BOOL bAInfo, CHAR *pcLabel)
{
    CHAR cNWMsg[256];
    CHAR *pcNWS = "";
    switch (dwNWS)
    {
        case NWS_NEW:
            pcNWS = "new";
            break;
        case NWS_FOUND:
            pcNWS = "found";
            break;
        case NWS_DELETE:
            pcNWS = "delete";
            break;
        default:
            break;
    }
    wsnprintf(cNWMsg, sizeof(cNWMsg), "%s(ad n=%lu id=%lx p=%08lx): nw %s
(n=%lu id=%lx p=%08lx)\n",
        pcLabel, pAdapter->dwIndex, pAdapter->dwID, pAdapter, pcNWS,
dwIndex, pNetwork->dwID, pNetwork);
    OutputDebugString(cNWMsg);
    if (bAInfo)
    {
        wsnprintf(cNWMsg, sizeof(cNWMsg), " adapter: type=%lu ipct=%lu
eth=%02x-%02x-%02x-%02x-%02x-%02x: ",
            pAdapter->dwIFType, pAdapter->dwIpCount,
            pAdapter->PhysAddr.Eth[0],
            pAdapter->PhysAddr.Eth[1],
            pAdapter->PhysAddr.Eth[2],
            pAdapter->PhysAddr.Eth[3],
            pAdapter->PhysAddr.Eth[4],
            pAdapter->PhysAddr.Eth[5]);
        OutputDebugString(cNWMsg);
        OutputDebugString(pAdapter->cDesc);
        OutputDebugString("\n");
    }
    PIP_INFO pInfo = &(pAdapter->IpInfo[dwIndex]);
    wsnprintf(cNWMsg, sizeof(cNWMsg), " ip: addr=%08lx mask=%08lx
flags=%08lx gw=%08lx\n",
        pInfo->dwAddr, pInfo->dwMask, pInfo->dwFlags, pInfo->dwGateway);
    OutputDebugString(cNWMsg);
    wsnprintf(cNWMsg, sizeof(cNWMsg), " nw: addr=%08lx mask=%08lx
flags=%08lx refct=%0lu status=%lu gw=%02x-%02x-%02x-%02x-%02x-%02x: ",
        pNetwork->dwNetAddr, pNetwork->dwNetMask, pNetwork->dwNetFlags,
pNetwork->dwRefCount, pNetwork->dwNWStatus,
        pNetwork->GWPhysAddr.Eth[0],

```

"C:\WINDOWS" 009000

```

        pNetwork->GWPhysAddr.Eth[1],
        pNetwork->GWPhysAddr.Eth[2],
        pNetwork->GWPhysAddr.Eth[3],
        pNetwork->GWPhysAddr.Eth[4],
        pNetwork->GWPhysAddr.Eth[5]);
    OutputDebugString(cNWSMsg);
    OutputDebugString(pNetwork->cName);
    OutputDebugString("\n");
}

#endif

DWORD VR_ADAPTER::AddNetworks(PVR_COMPUTER pComputer)
{
    DWORD j;
    DWORD dwRes = 0;
    PVR_NETWORK pNetwork;
    IP_NETWORK IPNetwork = {0};
    PVR_NETWORK NewNetworks[MAX_IP_ADDR] = {0};
#ifdef _NETWORK_DEBUG
    CHAR cNWSMsg[256];
    DWORD dwNWS;
    BOOL bAInfo = TRUE;
#endif

    // decrement network ref counts; if still used, they will be incremented
back
    for (j = 0; j < MAX_IP_ADDR; j++)
    {
        pNetwork = Networks[j];
        if (pNetwork && pNetwork->dwRefCount)
            pNetwork->dwRefCount--;
    }

    // add networks
    for (j = 0; j < dwIpCount; j++)
    {
        memset(&IPNetwork, 0, sizeof(IPNetwork));
        // if (IpInfo[j].dwGateway)
        //     IPNetwork.dwNetAddr = IpInfo[j].dwGateway &
IpInfo[j].dwMask;
        // else
            IPNetwork.dwNetAddr = IpInfo[j].dwAddr & IpInfo[j].dwMask;
        if (!(IPNetwork.dwNetAddr))
            continue;    //break;    //?

#ifdef _NETWORK_DEBUG
        dwNWS = NWS_NEW;
#endif

        IPNetwork.dwNetFlags = NWF_NET_ADDR;
        IPNetwork.dwNetMask = IpInfo[j].dwMask;
        if (dwAdapterFlags & ADAPTER_INFO_DIALUP)
        {
            // for now
            memcpy(&(IPNetwork.GWPhysAddr), &PhysAddr,
sizeof(IPNetwork.GWPhysAddr));
            IPNetwork.dwNetFlags = NWF_PHYS_ADDR;
        }
    }
}

```



```

        pNetwork = Networks[j];
        if (pNetwork && !(pNetwork->dwRefCount))
        {
#ifdef _NETWORK_DEBUG
            DumpNetworkInfo(this, j, pNetwork, NWS_DELETE, bAInfo,
"AddNetworks");
            bAInfo = FALSE;
#endif
            dwRes |= ADAPTER_NETWORK_DELETED;
            if (pComputer->dwNetCount)
                pComputer->dwNetCount--;
            if (pComputer->dwNetCountLAN && pNetwork->dwNWStatus)
                pComputer->dwNetCountLAN--;
            pNetwork->DestroyObject();
            Networks[j] = NULL;
        }
        Networks[j] = NewNetworks[j];
    }

    return dwRes;
}

BOOL VR_ADAPTER::DelNetworks(PVR_COMPUTER pComputer)
{
    PVR_NETWORK pNetwork;
    BOOL bChanged = FALSE;
#ifdef _NETWORK_DEBUG
    BOOL bAInfo = TRUE;
#endif
    for (DWORD j = 0; j < dwIpCount; j++)
    {
        pNetwork = Networks[j];
        if (pNetwork)
        {
            Networks[j] = NULL;
            if (pNetwork->dwRefCount)
            {
                if (pNetwork->dwRefCount == 1)
                {
#ifdef _NETWORK_DEBUG
                    DumpNetworkInfo(this, j, pNetwork, NWS_DELETE,
bAInfo, "DelNetworks");
                    bAInfo = FALSE;
#endif
                    bChanged = TRUE;
                    if (pComputer->dwNetCount)
                        pComputer->dwNetCount--;
                    if (pComputer->dwNetCountLAN &&
pNetwork->dwNWStatus)
                        pComputer->dwNetCountLAN--;
                    pNetwork->DestroyObject();
                }
                else
                    pNetwork->dwRefCount--;
            }
        }
    }
}

```



```

CHAR InfoBuffer[1000];
DWORD dwInfoSize = sizeof(InfoBuffer);
if (GetTokenInformation(hAccessToken, TokenUser,
InfoBuffer, sizeof(InfoBuffer), &dwInfoSize))
{
    PTOKEN_USER pTokenUser = (PTOKEN_USER)
InfoBuffer;

    CHAR cAccountName[200];
    DWORD dwAccountSize = sizeof(cAccountName);
    DWORD dwDomainSize = sizeof(cDomainName);
    SID_NAME_USE snu;
    LookupAccountSid(NULL,
                                pTokenUser->User.Sid,
                                cAccountName,
                                &dwAccountSize,
                                cDomainName,
                                &dwDomainSize,
                                &snu);
}
CloseHandle(hAccessToken);
}
}
if (!cDomainName[0])
    strcpy(cDomainName, "Local");

char cComputerName[MAX_COMPUTERNAME_LENGTH + 1];
DWORD dwLen = sizeof(cComputerName);
if (!GetComputerName(cComputerName, &dwLen))
    strcpy(cComputerName, "Unknown");

wsnprintf(buf, cbSize, "%s%s:%s", cPlatform, cDomainName,
cComputerName);
}

BOOL WINAPI GetAdapterCID(PIP_ADAPTER pAdapter, CHAR *pChar, DWORD dwLen)
{
    CHAR cType[16];
    switch(pAdapter->dwIFType)
    {
        case MIB_IF_TYPE_OTHER:
            wsnprintf(cType, sizeof(cType), "Other");
            break;
        case MIB_IF_TYPE_ETHERNET:
            wsnprintf(pChar, dwLen,
"ETHER%02x-%02x-%02x-%02x-%02x-%02x",
                pAdapter->PhysAddr.Eth[0],
                pAdapter->PhysAddr.Eth[1],
                pAdapter->PhysAddr.Eth[2],
                pAdapter->PhysAddr.Eth[3],
                pAdapter->PhysAddr.Eth[4],
                pAdapter->PhysAddr.Eth[5]);
            return TRUE;
        case MIB_IF_TYPE_TOKENRING:
            wsnprintf(pChar, dwLen,
"TOKEN%02x-%02x-%02x-%02x-%02x",
                pAdapter->PhysAddr.Eth[0],

```



```

/*****
**                               Zone Labs Monitor                               **
**                               Copyright(c) Zone Labs, 2001                       **
*****/

/*
    vscmpclient.cpp (vsmon.exe)

    Handles client side of Client Monitoring Protocol (CMP)

*/

#include <windows.h>
#include <winsock.h>
#include <time.h>
#include "vsutil.h"
#include "firewall.h"
#include "vsmon.h"
#include "vsruleapi.h"
#include "rsaapi.h"
#include "lsmgr.h"

extern HMODULE hDataModule;

#define ORDINAL char *

// encryption
#define PRIVATEENCRYPTNOHEADER 11
#define PRIVATEDECRYPTNOHEADER 12
typedef int (*PPRIVATE_ENCRYPT) (unsigned char *, unsigned int *, unsigned
char *, unsigned int, R_RSA_PRIVATE_KEY *);
typedef int (*PPRIVATE_DECRYPT) (unsigned char *, unsigned int *, unsigned
char *, unsigned int, R_RSA_PRIVATE_KEY *);
PPRIVATE_ENCRYPT pEncrypt = NULL;
PPRIVATE_DECRYPT pDecrypt = NULL;
BOOL bEncrypt = TRUE;

// license key
#define CMP_PREFERRED_LICMODE(m) ((m) == lmUnrestricted) || ((m) ==
lmSubscription)
#define CMP_LICKEY_TIME (12 * 60 * 60 * 1000)
#define LICINSTALLLICENSEBUF 6
#define LICGETPARSFROMBUF 13
LPFN_LICINSTALLLICENSEBUF pInstallLicense = NULL;
LPFN_LICGETPARSFROMBUF pGetLicPars = NULL;

// antivirus
// #define CMP_AV_ENABLE
#define CMP_AVKEY_TRENDMICRO "SOFTWARE\\TrendMicro\\PC-cillin"
#define CMP_AVNAME_TRENDMICRO "PNTIMON.EXE"

// rule to allow CMP traffic
typedef struct {
    FW_RULE_HEADER FwRule;

```



```

// -----
// Utilities
// -----

DWORD WINAPI CMPCThread(LPVOID lpPar)
{
    PCMP_CLIENT pClient = (PCMP_CLIENT) lpPar;
    return pClient->Execute();
}

VOID WINAPI SwapPktCommon(PPKT_COMMON pPkt, BOOL bToNet = TRUE)
{
    if (bToNet)
    {
        pPkt->wPacketID = htons(pPkt->wPacketID);
        pPkt->wPacketSize = htons(pPkt->wPacketSize);
        pPkt->wProtocolVersion = htons(pPkt->wProtocolVersion);
        pPkt->wPacketOptions = htons(pPkt->wPacketOptions);
        pPkt->dPacketCRC = htonl(pPkt->dPacketCRC);
        // pPkt->dSendIPAddress stays in network byte order
        pPkt->dSendProductID = htonl(pPkt->dSendProductID);
        pPkt->SendVersion.dVers[0] = htonl(pPkt->SendVersion.dVers[0]);
        pPkt->SendVersion.dVers[1] = htonl(pPkt->SendVersion.dVers[1]);
    }
    else
    {
        pPkt->wPacketID = ntohs(pPkt->wPacketID);
        pPkt->wPacketSize = ntohs(pPkt->wPacketSize);
        pPkt->wProtocolVersion = ntohs(pPkt->wProtocolVersion);
        pPkt->wPacketOptions = ntohs(pPkt->wPacketOptions);
        pPkt->dPacketCRC = ntohl(pPkt->dPacketCRC);
        // pPkt->dSendIPAddress stays in network byte order
        pPkt->dSendProductID = ntohl(pPkt->dSendProductID);
        pPkt->SendVersion.dVers[0] = ntohl(pPkt->SendVersion.dVers[0]);
        pPkt->SendVersion.dVers[1] = ntohl(pPkt->SendVersion.dVers[1]);
    }
}

VOID WINAPI SwapClientHello(PCL_HELLO_PACKET pPkt)
{
    SwapPktCommon((PPKT_COMMON) pPkt);
    pPkt->dReserved = htonl(pPkt->dReserved);
}

VOID WINAPI SwapClientResponse(PCL_CHALLENGE_RESPONSE pPkt)
{
    SwapPktCommon((PPKT_COMMON) pPkt);
    pPkt->dRouterSessionID = htonl(pPkt->dRouterSessionID);
    pPkt->dTimestamp = htonl(pPkt->dTimestamp);
    pPkt->dStatus = htonl(pPkt->dStatus);
    pPkt->dReserved = htonl(pPkt->dReserved);
}

```

```

VOID WINAPI SwapRouterChallenge(PRT_CHALLENGE_PACKET pPkt)
{
    SwapPktCommon((PPKT_COMMON) pPkt, FALSE);
    pPkt->dRouterSessionID = ntohl(pPkt->dRouterSessionID);
    pPkt->dResponseTime = ntohl(pPkt->dResponseTime);
    pPkt->dTimestamp = ntohl(pPkt->dTimestamp);
    pPkt->dReserved = ntohl(pPkt->dReserved);
}

VOID WINAPI SwapOptCommon(POPT_COMMON pOpt, BOOL bToNet = FALSE)
{
    if (bToNet)
    {
        pOpt->wOptionID = htons(pOpt->wOptionID);
        pOpt->wOptionSize = htons(pOpt->wOptionSize);
    }
    else
    {
        pOpt->wOptionID = ntohs(pOpt->wOptionID);
        pOpt->wOptionSize = ntohs(pOpt->wOptionSize);
    }
}

VOID WINAPI SwapOptClientVersion(POPT_CLIENT_VERSION pOpt)
{
    SwapOptCommon((POPT_COMMON) pOpt);
    pOpt->dProductID = ntohl(pOpt->dProductID);
    pOpt->ProductVersion.dVers[0] = ntohl(pOpt->ProductVersion.dVers[0]);
    pOpt->ProductVersion.dVers[1] = ntohl(pOpt->ProductVersion.dVers[1]);
}

VOID WINAPI SwapOptLicense(POPT_ZL_LICENSE pOpt)
{
    SwapOptCommon((POPT_COMMON) pOpt);
    pOpt->wNumberOfUsers = ntohs(pOpt->wNumberOfUsers);
}

VOID WINAPI SwapOptUserPrompt(POPT_USER_PROMPT pOpt)
{
    SwapOptCommon((POPT_COMMON) pOpt);
}

VOID WINAPI SwapOptAntiVirus(POPT_ANTI_VIRUS pOpt)
{
    SwapOptCommon((POPT_COMMON) pOpt);
    pOpt->dProductID = ntohl(pOpt->dProductID);
    pOpt->AVVersion.dVers[0] = ntohl(pOpt->AVVersion.dVers[0]);
    pOpt->AVVersion.dVers[1] = ntohl(pOpt->AVVersion.dVers[1]);
    pOpt->bAutoUpdate = ntohl(pOpt->bAutoUpdate);
    pOpt->bRealTimeMonitor = ntohl(pOpt->bRealTimeMonitor);
    pOpt->dReserved1 = ntohl(pOpt->dReserved1);
    pOpt->dReserved2 = ntohl(pOpt->dReserved2);
}

```

```

INT WINAPI CompareVersions(PCMPVERS pVers1, PCMPVERS pVers2)
{
    for (INT i = 0; i < 4; i++)
    {
        INT iDiff = (INT) pVers1->wVers[i] - pVers2->wVers[i];
        if (iDiff)
            return (iDiff < 0) ? -1 : 1;
    }
    return 0;
}

```

```

BOOL WINAPI ParseVersion(CHAR *pcVersStr, PCMPVERS pVers)
{
    if (pcVersStr && (*pcVersStr))
    {
        INT i = 0;
        CHAR *pStr = pcVersStr;
        CHAR *pDot = strchr(pStr, '.');
        while (pDot)
        {
            *pDot = 0;
            pVers->wVers[i] = (WORD) atoi(pStr);
            *pDot = '.';
            pStr = pDot + 1;
            if (++i > 3)
            {
                pStr = NULL;
                break;
            }
            pDot = strchr(pStr, '.');
        }
        if (pStr)
            pVers->wVers[i] = (WORD) atoi(pStr);
        if (pVers->dVers[0] || pVers->dVers[1])
            return TRUE;
    }
    return FALSE;
}

```

```

// -----
// CMP_CLIENT
// -----

```

```

CMP_CLIENT::CMP_CLIENT(
    DWORD dwNetworkId,
    DWORD dwNWMonStatus,
    LPVOID lpKey,
    DWORD dwGWType,
    DWORD dwClientType,
    PCMPVERS pClientVers,
    DWORD dwGatewayAddr,
    DWORD dwClientAddr,
    WORD wPort,
    DWORD dwFlags)

```

```

{
    this->dwNetworkId = dwNetworkId;
    this->dwNWMonStatus = dwNWMonStatus;
    this->lpKey = lpKey;
    dwGatewayType = dwGWType;
    this->dwClientType = dwClientType;
    if (pClientVers)
    {
        ClientVersion.dVers[0] = pClientVers->dVers[0];
        ClientVersion.dVers[1] = pClientVers->dVers[1];
    }
    this->dwGatewayAddr = dwGatewayAddr;
    this->dwClientAddr = dwClientAddr;
    this->wPort = wPort;
    this->dwFlags = dwFlags;
    hThread = NULL;
    sock = INVALID_SOCKET;
    dwStatus = -1;
    dwAVProcId = 0;
    dwLicKeyStatus = LKS_UNKNOWN;
    dwLicKeyTime = 0;
    memset(cLicKey, 0, sizeof(cLicKey));

#ifdef _CMPC_DEBUG
    wsnprintf(cCMPCMsg, sizeof(cCMPCMsg),
        "CMP_CLIENT(%x): t=%x nw(id=%x st=%lu) gw(t=%x a=%08lx k=%x)
        cl(t=%x v=%hu.%hu.%hu a=%08lx) p=%hu\n",
        this, GetTickCount(), dwNetworkId, dwNWMonStatus, dwGWType,
        dwGatewayAddr, lpKey,
        dwClientType, ClientVersion.wVers[0], ClientVersion.wVers[1],
        ClientVersion.wVers[2], ClientVersion.wVers[3],
        dwClientAddr, ntohs(wPort));
    OutputDebugString(cCMPCMsg);
#endif

    // validate parameters (other checks?)
    if (!(dwNetworkId && lpKey && pClientVers &&
        dwGatewayType && dwClientType && (ClientVersion.dVers[0] ||
        ClientVersion.dVers[1]) &&
        dwGatewayAddr && dwClientAddr && wPort) ||
        (dwNWMonStatus == NWMS_EXEMPT))
    {
#ifdef _CMPC_DEBUG
        OutputDebugString("CMP_CLIENT error: bad parameters\n");
#endif
        return;
    }

    // create socket
    sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if (sock != INVALID_SOCKET)
    {
        struct sockaddr_in addr;
        memset(&addr, 0, sizeof(addr));
        addr.sin_family = AF_INET;
        addr.sin_port = wPort;
        addr.sin_addr.s_addr = dwClientAddr;
        if (bind(sock, (const struct sockaddr *) &addr, sizeof(addr)))

```

```

        {
            closesocket(sock);
            sock = INVALID_SOCKET;
        }
        else
        {
//            INT iTimeout = CMPC_SEND_TIMEOUT;
//            setsockopt(sock, SOL_SOCKET, SO_SNDTIMEO, (char *)
&iTimeout, sizeof(iTimeout));
        }
    }
    if (sock == INVALID_SOCKET)
    {
#ifdef _CMPC_DEBUG
        wsnprintf(cCMPCMsg, sizeof(cCMPCMsg), "CMP_CLIENT error: socket
(%d)\n", WSAGetLastError());
        OutputDebugString(cCMPCMsg);
#endif
        return;
    }

    // start main thread
    DWORD dwThreadId;
    dwStatus = 0;
    hThread = CreateThread(NULL, 0x4000, CMPCThread, this, 0, &dwThreadId);
    if (!hThread)
    {
#ifdef _CMPC_DEBUG
        OutputDebugString("CMP_CLIENT error: unable to create thread\n");
#endif
        dwStatus = -1;
        return;
    }
}

CMP_CLIENT::~CMP_CLIENT()
{
#ifdef _CMPC_DEBUG
    wsnprintf(cCMPCMsg, sizeof(cCMPCMsg), "~CMP_CLIENT(%x): t=%x\n", this,
GetTickCount());
    OutputDebugString(cCMPCMsg);
#endif
    dwStatus = -1;
    if (sock != INVALID_SOCKET)
    {
        SOCKET tmp_sock = (SOCKET) InterlockedExchange((LPLONG) &sock,
INVALID_SOCKET);
        closesocket(tmp_sock);
    }
    if (hThread)
    {
        if (WaitForSingleObject(hThread, CMPC_STOP_WAIT) == WAIT_TIMEOUT)
        {
#ifdef _CMPC_DEBUG
            wsnprintf(cCMPCMsg, sizeof(cCMPCMsg), "~CMP_CLIENT: thread
timeout (%08lx)\n", hThread);
            OutputDebugString(cCMPCMsg);

```

```

#endif
        SuspendThread(hThread);
        TerminateThread(hThread, 0);
    }
    CloseHandle(hThread);
    hThread = NULL;
}

VOID CMP_CLIENT::SetupPktCommon(PPKT_COMMON pPkt, WORD wId, WORD wSize)
//
// Sets up common message header
//
{
    pPkt->wPacketID = wId;
    pPkt->wPacketSize = wSize;
    pPkt->wProtocolVersion = CMP_VERSION;
    pPkt->wPacketOptions = 0;
    pPkt->dPacketCRC = 0;
    pPkt->dSendIPAddress = dwClientAddr;
    pPkt->dSendProductID = dwClientType;
    pPkt->SendVersion.dVers[0] = ClientVersion.dVers[0];
    pPkt->SendVersion.dVers[1] = ClientVersion.dVers[1];
}

VOID CMP_CLIENT::SetupHello(PCL_HELLO_PACKET pPkt)
//
// Sets up hello message
//
{
    SetupPktCommon((PPKT_COMMON) pPkt, CMPM_CLIENT_HELLO,
sizeof(CL_HELLO_PACKET));
    pPkt->dReserved = 0;
    SwapClientHello(pPkt);
    pPkt->dPacketCRC = CheckSum((USHORT *) pPkt, (INT)
ntohs(pPkt->wPacketSize));
}

VOID CMP_CLIENT::SetupResponse(PCL_CHALLENGE_RESPONSE pPkt,
PRT_CHALLENGE_PACKET pChal)
//
// Sets up response message
//
{
    SetupPktCommon((PPKT_COMMON) pPkt, CMPM_CLIENT_RESPONSE,
sizeof(CL_CHALLENGE_RESPONSE));
    pPkt->dRouterSessionID = pChal->dRouterSessionID;
    pPkt->dTimestamp = pChal->dTimestamp;
    pPkt->dStatus = dwStatus;
    if (dwLicKeyStatus == LKS_ACCEPTED)
        pPkt->dStatus |= CMPR_LICENSE_ACCEPTED;
    pPkt->dReserved = 0;
    SwapClientResponse(pPkt);
    pPkt->dPacketCRC = CheckSum((USHORT *) pPkt, (INT)
ntohs(pPkt->wPacketSize));
}

```

```

}

BOOL CMP_CLIENT::Encrypt(UCHAR *pucIn, UCHAR *pucOut, DWORD dwLen)
//
// Encrypts packet for sending to gateway
//
{
    if (bEncrypt)
    {
        if (!pEncrypt)
        {
            if (hDataModule)
                pEncrypt = (PPRIVATE_ENCRYPT)
GetProcAddress(hDataModule, (ORDINAL) PRIVATEENCRYPTNOHEADER);
            if (!pEncrypt)
                return FALSE;
        }

        UINT uiOutLen;
        UCHAR *pucCurIn = pucIn;
        UCHAR *pucCurOut = pucOut;
        DWORD dwCount = (dwLen + CMP_KEYLENGTH - 1) / CMP_KEYLENGTH;
        for (DWORD i = 0; i < dwCount; i++)
        {
            uiOutLen = CMP_KEYLENGTH;
            pEncrypt(pucCurOut, &uiOutLen, pucCurIn, CMP_KEYLENGTH,
(R_RSA_PRIVATE_KEY *) lpKey);
            pucCurIn += CMP_KEYLENGTH;
            pucCurOut += CMP_KEYLENGTH;
        }
    }
    else
        memcpy(pucOut, pucIn, dwLen);
    return TRUE;
}

BOOL CMP_CLIENT::Decrypt(UCHAR *pucIn, UCHAR *pucOut, DWORD dwLen)
//
// Decrypts packet from gateway
//
{
    if (bEncrypt)
    {
        if (!pDecrypt)
        {
            if (hDataModule)
                pDecrypt = (PPRIVATE_DECRYPT)
GetProcAddress(hDataModule, (ORDINAL) PRIVATEDECRYPTNOHEADER);
            if (!pDecrypt)
                return FALSE;
        }

        UINT uiOutLen;
        UCHAR *pucCurIn = pucIn;
        UCHAR *pucCurOut = pucOut;
        DWORD dwCount = (dwLen + CMP_KEYLENGTH - 1) / CMP_KEYLENGTH;
    }
}

```

```

        for (DWORD i = 0; i < dwCount; i++)
        {
            uiOutLen = CMP_KEYLENGTH;
            pDecrypt(pucCurOut, &uiOutLen, pucCurIn, CMP_KEYLENGTH,
(R_RSA_PRIVATE_KEY *) lpKey);
            pucCurIn += CMP_KEYLENGTH;
            pucCurOut += CMP_KEYLENGTH;
        }
    }
    else
        memcpy(pucOut, pucIn, dwLen);
    return TRUE;
}

BOOL CMP_CLIENT::DoSend(LPVOID pData, DWORD dwSize)
//
// Sends CMP message to gateway
//
{
    struct sockaddr_in remAddr;
    memset(&remAddr, 0, sizeof(remAddr));
    remAddr.sin_family = AF_INET;
    remAddr.sin_port = wPort;
    remAddr.sin_addr.s_addr = dwGatewayAddr;

    for (INT i = 0; i < CMPC_SEND_TRIES; i++)
    {
        INT iLen = sendto(sock, (const char *) pData, (INT) dwSize, 0,
(const struct sockaddr *) &remAddr, sizeof(remAddr));
        if (iLen == SOCKET_ERROR)
        {
            if (!WSAGetLastError())
                return FALSE;
        }
        else if (iLen == (INT) dwSize)
        {
            return TRUE;
        }
    }

    return FALSE;
}

BOOL CMP_CLIENT::SendHello(VOID)
//
// Sends hello message to gateway
//
{
#ifdef _CMPC_DEBUG
    wsnprintf(cMPCMsg, sizeof(cMPCMsg), "CMP_CLIENT::SendHello (%08lx):
t=%x\n", this, GetTickCount());
    OutputDebugString(cMPCMsg);
#endif

    UCHAR ucInBuf[CMPC_MAX_HELLO] = {0};
    UCHAR ucOutBuf[CMPC_MAX_HELLO] = {0};
}

```



```

HKEY hKey = NULL;
CMPVERS AVVersion = {0};
CHAR cVersKey[256] = {0};
if (RegOpenKeyEx(HKEY_LOCAL_MACHINE, cKey, 0,
KEY_ENUMERATE_SUB_KEYS, &hKey) == ERROR_SUCCESS)
{
    INT i = 0;
    CHAR cVers[256];
    CMPVERS tmpVers;
    while (1)
    {
        cVers[0] = 0;
        if (RegEnumKey(hKey, i, cVers, sizeof(cVers)) !=
ERROR_SUCCESS)
            break;

        i++;
        memset(&tmpVers, 0, sizeof(tmpVers));
        if (ParseVersion(cVers, &tmpVers))
        {
            if (CompareVersions(&AVVersion, &tmpVers)
< 0)
            {
                memcpy(&AVVersion, &tmpVers,
sizeof(AVVersion));
                wsprintf(cVersKey, "%s%s", cKey,
cVers);
            }
        }
        RegCloseKey(hKey);
    }

    // no version found
    if (!cVersKey[0])
    {
        dwStatus = CMPR_AV_NOT_INSTALLED;
        return;
    }

    // check version
    if (pAVOpt->AVVersion.dVers[0] ||
pAVOpt->AVVersion.dVers[1])
    {
        if (CompareVersions(&AVVersion, &(pAVOpt->AVVersion))
< 0)
        {
            dwStatus = CMPR_WRONG_AV_VERSION;
            return;
        }
    }

    // check auto-update
    if (pAVOpt->bAutoUpdate)
    {
        lstrcat(cVersKey, "\\Autoupdate"); //?
        DWORD dwAutoUpdate =
GetRegistryValueDWord(HKEY_LOCAL_MACHINE, cVersKey, "Autoupdate", 0);

```

T09070 "E58000000"

```

        if (!dwAutoUpdate)
        {
            dwStatus = CMPR_AV_NO_AUTOUPDATE;
            return;
        }
    }

    // check real-time monitor
    if (pAVOpt->bRealTimeMonitor)
    {
        if (dwAVProcId)
        {
            // make sure it's still there (need to make sure
this works in 9x; consider using VSGetProcessInfo)
            char procName[MAX_PATH];
            if (VSGetProcessFileName(dwAVProcId, procName,
sizeof(procName)))
            {
                char baseName[MAX_PATH];
                ExtractFileBase(procName, baseName,
sizeof(baseName));
                if (lstrcmpi(baseName,
CMP_AVNAME_TRENDMICRO))
                    dwAVProcId = 0;
            }
            else
                dwAVProcId = 0;
        }
        if (!dwAVProcId)
            dwAVProcId =
GetModulePIDEx(CMP_AVNAME_TRENDMICRO, IMRX_BASE_ONLY);
        if (!dwAVProcId)
        {
            dwStatus = CMPR_AV_NO_REALTIME_MONITOR;
            return;
        }
    }
    break;
}
default:
    break;
}
}
}

```

```

VOID CMP_CLIENT::ValidateLicenseKey(POPT_ZL_LICENSE pZLOpt)
//
// Validates client license key
//
{
    if (!pGetLicPars)
    {
        if (hDataModule)
            pGetLicPars = (LPFN_LICGETPARSFROMBUF)
GetProcAddress(hDataModule, (ORDINAL) LICGETPARSFROMBUF);
        if (!pGetLicPars)
            return;
    }
}

```

```

LIC_ERROR licErr;
LICENSE_PARAMETERS licPars = {0};

// check for existing license
CHAR cKey[32] = {0};
if (GetRegistryValueString(HKEY_LOCAL_MACHINE, KEY_REGISTRATION,
"SerialNum", cKey, sizeof(cKey)) && cKey[0])
{
    licErr = pGetLicPars(cKey, &licPars);
    if ((licErr == lserr_OK) &&
CMP_PREFERRED_LICMODE(licPars.licenseMode))
    {
        // already have a (real) license
        dwLicKeyStatus = LKS_DECLINED;
        return;
    }
}

// get info for router-specified key
licErr = pGetLicPars(pZLOpt->cLicenseKey, &licPars);
if (licErr != lserr_OK)
{
    dwStatus = CMPR_INVALID_ZL_LICENSE;
    return;
}

// not a router license key
if (licPars.licenseMode != lmGateway)
{
    dwStatus = CMPR_INVALID_ZL_LICENSE;
    return;
}

// bad oem code
DWORD dwGWType = OEMCodeToGWType(licPars.oemCode);
if (dwGWType != dwGatewayType)
{
    dwStatus = CMPR_INVALID_ZL_LICENSE;
    return;
}

// too many concurrent users
if (pZLOpt->wNumberOfUsers >= licPars.useCount)
{
    dwStatus = CMPR_EXCEED_ZL_LICENSE;
    return;
}

// we'll take it (or at least try)
if (!pInstallLicense)
{
    if (hDataModule)
        pInstallLicense = (LPFN_LICINSTALLLICENSEBUF)
GetProcAddress(hDataModule, (ORDINAL) LICINSTALLLICENSEBUF);
    if (!pInstallLicense)
        return;
}
licErr = pInstallLicense(pZLOpt->cLicenseKey);

```

FOR "E" SERIES

```

switch (licErr)
{
    case lserr_OK:
    case lserr_AlreadyInstalled:
        dwLicKeyStatus = LKS_ACCEPTED;
        dwLicKeyTime = GetTickCount();
        memcpy(cLicKey, pZLOpt->cLicenseKey, sizeof(cLicKey));
        break;
    default:
        dwStatus = CMPR_INVALID_ZL_LICENSE; ///?
        break;
}
}

```

```

BOOL CMP_CLIENT::Validate(PRT_CHALLENGE_PACKET pChal, DWORD dwLen)
//
// Validates client
//
{
    // check packet size
    WORD wPktSize = ntohs(pChal->wPacketSize);
    if ((wPktSize < sizeof(RT_CHALLENGE_PACKET)) || (wPktSize >
CMP_MAX_CHALLENGE) || (wPktSize > dwLen))
    {
        dwStatus = CMPC_BAD_PACKET;
        return FALSE;
    }

    // check checksum
    WORD wChkSum = CheckSum((USHORT *) pChal, (INT) wPktSize);
    if (wChkSum)
    {
        dwStatus = CMPC_BAD_PACKET;
        return FALSE;
    }

    // assume ok
    dwStatus = 0;
    SwapRouterChallenge(pChal);
    // check version, other header values?

    // check options
    if (pChal->wPacketOptions)
    {
        WORD wOptSize;
        WORD wSize = sizeof(RT_CHALLENGE_PACKET);
        POPT_COMMON pOpt = (POPT_COMMON) &(pChal[1]);
        for (WORD i = 0; i < pChal->wPacketOptions; i++)
        {
            wOptSize = ntohs(pOpt->wOptionSize);
            if (wOptSize < sizeof(OPT_COMMON))
            {
                dwStatus = CMPC_BAD_PACKET;
                return FALSE;
            }
            wSize += wOptSize;
            if (wSize > wPktSize)

```

FOR "E" FILES


```

        }
        pOpt = (POPT_COMMON) ((CHAR *) pOpt + wOptSize);
    }
}

return TRUE;
}

DWORD CMP_CLIENT::Execute(VOID)
//
// Processes router challenges
//
{
    // seed random number generator
    srand((unsigned) time(NULL));

    // add fw rule to allow client/router communication
    HANDLE hRule = NULL;
    PFW_RULE_CMP pRule = (PFW_RULE_CMP) &AllowCMP;
    pRule->FwRuleCSockIn.dwAddr1 = dwClientAddr;
    pRule->FwRuleCSockIn.wPort1 = wPort;
    pRule->FwRuleGSockIn.dwAddr1 = dwGatewayAddr;
    pRule->FwRuleGSockIn.wPort1 = wPort;
    pRule->FwRuleCSockOut.dwAddr1 = dwClientAddr;
    pRule->FwRuleCSockOut.wPort1 = wPort;
    pRule->FwRuleGSockOut.dwAddr1 = dwGatewayAddr;
    pRule->FwRuleGSockOut.wPort1 = wPort;
    FirewallAddRule(&AllowCMP, &hRule);

    // previously monitored - send hello (unless emergency lock is on)
    if (dwNWMonStatus == NWMS_MONITORED)
    {
        DWORD dwFWLock = FirewallGetLock();
        if ((dwFWLock != LOCKLEV_EMERG) && (dwFWLock != LOCKLEV_EMERG_MGR))
            SendHello();
    }

    // listen for router challenges
    INT iLen;
    INT iSize;
    DWORD dwWait;
    CHAR cBuf[CMPC_MAX_CHALLENGE];
    struct sockaddr_in remAddr;
    while (dwStatus != -1)
    {
        memset(cBuf, 0, sizeof(cBuf));
        memset(&remAddr, 0, sizeof(remAddr));
        iSize = sizeof(remAddr);
        iLen = recvfrom(sock, cBuf, sizeof(cBuf), 0, (struct sockaddr *)
&remAddr, &iSize);
        if ((dwStatus != -1) &&
            (remAddr.sin_addr.s_addr == dwGatewayAddr) &&
            (remAddr.sin_port == wPort) &&
            (iLen >= sizeof(RT_CHALLENGE_PACKET)))
        {
#ifdef _CMPC_DEBUG
            wsprintf(cMPCMsg, sizeof(cMPCMsg), "CMP_CLIENT::Execute
(%08lx): recvfrom t=%x l=%d\n",

```

"F090" E0000000

```

        this, GetTickCount(), iLen);
    OutputDebugString(cCMPMsg);
#endif

    UCHAR ucBuf[ CMP_MAX_CHALLENGE ] = {0};
    PRT_CHALLENGE_PACKET pChal = (PRT_CHALLENGE_PACKET) ucBuf;
    BOOL bRes = Decrypt((UCHAR *) cBuf, ucBuf, (DWORD) iLen);
    if (bRes)
    {
        if (ntohs(pChal->wPacketID) == CMPM_ROUTER_CHALLENGE)
            bRes = Validate(pChal, (DWORD) iLen);
        else
            bRes = FALSE;
    }
    if (bRes)
    {
        // respond after random delay
        if (pChal->dResponseTime)
        {
            dwWait = (((DWORD) rand()) %
                (pChal->dResponseTime * 1000)) + 1;
            Sleep(dwWait);
        }
        SendResponse(pChal);
        if (dwNWMonStatus != NWMS_MONITORED)
        {
            // remember that this network is monitored
            if (MonitorSetDWordValue(dwNetworkId,
                GDV_MONSTATUS, NWMS_MONITORED) != -1)
                dwNWMonStatus = NWMS_MONITORED;
        }
        if ((dwLicKeyStatus == LKS_ACCEPTED) &&
            pInstallLicense && cLicKey[0])
        {
            // check last license key update
            DWORD dwCurTime = GetTickCount();
            DWORD dwTDiff = (dwCurTime > dwLicKeyTime) ?
                dwCurTime - dwLicKeyTime : dwLicKeyTime - dwCurTime;
            if (dwTDiff > CMP_LICKEY_TIME)
            {
                // reinstall license key
                LIC_ERROR licErr =
                    pInstallLicense(cLicKey);

                switch (licErr)
                {
                    case lserr_OK:
                    case lserr_AlreadyInstalled:
                        dwLicKeyTime = dwCurTime;
                        break;
                    default:
                        break;
                }
            }
        }
    }
}

// delete fw rule

```

FOR "FOR" FILES


```
/*      Name: CMPAuthorizeTraffic
*      Purpose: determine whether or not to allow client traffic based on
status & remote port
*      return type: DWORD, 0 if traffic allowed; above zero: port specifying
redirect (host byte order); -1: block
*      Parameters:
*          WORD wPort, input, remote port (host byte order)
*          DWORD dwStatus, input, status from client table (set by
AnalyseDecryptedPacket)
*          DWORD dwCurrentTime, input, current timestamp
*          DWORD dwTimeout, input, max allowed time since last valid response
*/
DWORD CMPAuthorizeTraffic(WORD wPortDst, DWORD dwStatus, DWORD dwCurrentTime,
DWORD dwTimeout);
#endif /* _ZL_RTUTILS_H_ */
```

FORN "E59E0E09

```

/*****
/**      Zone Labs Client Monitoring Protocol Implementation      **/
/*****
/*
RTMAIN.C - router side main support routines

```

This Software, API structure and source code are Zone Labs confidential and proprietary information. No license or title is granted to you to the source code, API or intellectual property of the Software. Software use rights are only granted through Zone Labs' license agreement. Zone Labs reserves all rights not expressly granted herein.

Copyright(c) 2001 Zone Labs, Inc. All rights reserved.

```

*/
#include "rsaapi.h"
#include "vscmp.h"
#include "rtcmp.h"
#include "vsutils.h"
#include <memory.h>
#include <string.h>

#define DO_ENCRYPT_DECRYPT 1

BOOL CMPCreateChallengePacket(UCHAR *packetChallenge, DWORD dwBufLen,
                              DWORD dwProductID, DWORD dwRouterIPAddress, UCHAR
                              *pRouterVersion,
                              DWORD dwRouterSessionID, DWORD dwResponseTime,
                              DWORD dwTimestamp)
{
    PRT_CHALLENGE_PACKET pTempChallenge;
    PCMPVERS pTempVersion;

    if(dwBufLen<sizeof(RT_CHALLENGE_PACKET))
        return FALSE;
    pTempChallenge=(PRT_CHALLENGE_PACKET)packetChallenge;
    pTempChallenge->wPacketID=CMPM_ROUTER_CHALLENGE;
    pTempChallenge->wPacketSize=sizeof(RT_CHALLENGE_PACKET);
    pTempChallenge->wProtocolVersion=CMP_VERSION;
    pTempChallenge->wPacketOptions=0;
    pTempChallenge->dPacketCRC=0; /* checksum*/

    pTempChallenge->dSendIPAddress=dwRouterIPAddress; /* router ip address*/
    pTempChallenge->dSendProductID=dwProductID; /* router product
code*/

    pTempVersion=(PCMPVERS)pRouterVersion;
    pTempChallenge->SendVersion.dVers[0] = pTempVersion->dVers[0];
    pTempChallenge->SendVersion.dVers[1] = pTempVersion->dVers[1];

    pTempChallenge->dRouterSessionID=dwRouterSessionID; /* current session
id*/
    pTempChallenge->dResponseTime=dwResponseTime; /* expect response
within this time (seconds)*/
    pTempChallenge->dTimestamp=dwTimestamp; /* packet timestamp*/
    pTempChallenge->dReserved=0;
    return TRUE;
}

```

"ESSE" FILE


```

    }
    return rStatus;
}

/*
 * Name: CMPEncryptPacket
 * Purpose: Encrypt a Packet
 * return type: DWORD, >0 encrypted data length, 0:failed
 * Parameters:
 *     UCHAR *pucBufIn, input, a buffer containing data to be encrypted
 *     DWORD dwInputLen, input buffer length
 *     UCHAR *pucBufOut, output, a buffer holding data being encrypted
 *     DWORD dwBufLen, output buffer length
 */
DWORD CMPEncryptPacket(UCHAR *pucBufIn, DWORD dwInputLen, UCHAR *pucBufOut,
    DWORD dwBufLen)
{
#ifdef DO_ENCRYPT_DECRYPT
    int iLenControl;
    unsigned char* loopIn;
    unsigned char* loopOut;
    unsigned int tempOutLen;
#endif
    unsigned int tempInLen;

    tempInLen=(dwInputLen+15) & 0xffffffff0; //alignment /16*16
    if(dwBufLen<tempInLen)// output buffer too small
        return 0;
    if (!(pucBufIn && pucBufOut))
        return 0;

    memset(pucBufOut, 0, tempInLen);
#ifdef DO_ENCRYPT_DECRYPT
    loopIn=pucBufIn;
    loopOut=pucBufOut;
    iLenControl=tempInLen;
    while(iLenControl>15)
    {
        RSAPublicEncryptNoHeader(loopOut, &tempOutLen, loopIn, 16,
    &PublicKey);
        if(tempOutLen!=16)
            return 0;
        loopIn +=16;
        loopOut +=16;
        iLenControl-=16;
    }
#else
    memcpy(pucBufOut,pucBufIn,dwInputLen);
#endif
    return tempInLen;
}

/*
 * Name: CMPDecryptPacket
 * Purpose: Decrypt a Packet
 * return type: DWORD, >0 decrypted data length, 0:failed
 * Parameters:
 *     UCHAR *pucBufIn, input, a buffer containing data to be decrypted
 *     DWORD dwInputLen, input buffer length
 */

```

```

*          UCHAR *pucBufOut, output, a buffer holding data being decrypted
*          DWORD dwBufLen, output buffer length
*/
DWORD CMPDecryptPacket(UCHAR *pucBufIn, DWORD dwInputLen, UCHAR *pucBufOut,
DWORD dwBufLen)
{
#ifdef DO_ENCRYPT_DECRYPT
    int iLenControl;
    unsigned char* loopIn;
    unsigned char* loopOut;
    unsigned int tempOutLen;
#endif
    unsigned int tempInLen;

    tempInLen=(dwInputLen+15) & 0xffffffff0;//alignment /16*16
    if(tempInLen>dwBufLen)// output buffer too small
        return 0;
    if (!(pucBufIn && pucBufOut))
        return 0;

    memset(pucBufOut, 0, tempInLen);
#ifdef DO_ENCRYPT_DECRYPT
    loopIn=pucBufIn;
    loopOut=pucBufOut;
    iLenControl=tempInLen;
    while(iLenControl>15)
    {
        RSAPublicDecryptNoHeader(loopOut, &tempOutLen, loopIn, 16,
&PublicKey);
        if(tempOutLen!=16)
            return 0;
        loopIn+=16;
        loopOut+=16;
        iLenControl-=16;
    }
#else
    memcpy(pucBufOut,pucBufIn,dwInputLen);
#endif
    return tempInLen;
}

/*      Name: CMPAuthorizeTraffic
*      Purpose: determine whether or not to allow client traffic based on
status & remote port
*      return type: DWORD, 0 if traffic allowed; above 8080: port specifying
redirect (host byte order); -1: block
*      Parameters:
*          WORD wPort, input, remote port (host byte order)
*          DWORD dwStatus, input, status from client table (set by
AnalyseDecryptedPacket)
*          DWORD dwCurrentTime, input, current timestamp
*          DWORD dwTimeout, input, max allowed time since last valid response
*/
DWORD CMPAuthorizeTraffic(WORD wPortDst, DWORD dwStatus, DWORD dwCurrentTime,
DWORD dwTimeout)
{
    DWORD dwReturn;

```

```

if(dwStatus==CMPR_CLIENT_EXEMPT)
    dwReturn=0;
else if(dwStatus > 256 && (dwCurrentTime-dwStatus+256) < dwTimeout)
    dwReturn=0;
else if(wPortDst==80)//HTTP
{
    dwReturn=CMR_REDIRECT_BASEPORT;
    if(dwStatus==0)
        dwReturn += CMPR_NO_CLIENT_RESPONSE;           //no client response at
all
    else if(dwStatus<=256)
        dwReturn += dwStatus;                           //normal redirect
    else
        dwReturn += CMPR_CLIENT_TIMEOUT;                //timeout, (dwStatus >
256 && (dwCurrentTime-dwStatus-256) >= dwTimeout)
}
else//Other protocol
    dwReturn=-1;

return dwReturn;
}

```

T05020"ESSE0E0E0E

```

/*****
/**      Zone Labs Client Monitoring Protocol Test Program      **/
/*****
/*

```

ROUTER.C - Windows test program/router simulator

This Software, API structure and source code are Zone Labs confidential and proprietary information. No license or title is granted to you to the source code, API or intellectual property of the Software. Software use rights are only granted through Zone Labs' license agreement. Zone Labs reserves all rights not expressly granted herein.

Copyright(c) 2001 Zone Labs, Inc. All rights reserved.

```

*/
#include <windows.h>
#include <winsock.h>
#include <wsnwlinc.h>
#include "rsaapi.h"
#include "vscmp.h"
#include "rtcmp.h"
#include "vsutils.h"
#include <stdlib.h>
#include <stdio.h>
#include <memory.h>
#include <string.h>

// Broadcast addresses
#define SUBNET_BROADCAST_ADDR(addr, mask) (((addr) & (mask)) | (~(mask)))
#define SUBNET_MASK 0x00FFFFFF // network order
/*****
/**      Sample Client Status Table      **/
/**      if current router control table has IP address column **/
/**      please add one DWORD, whose 0-30 bits for client status **/
/**      and 31 bit for license status      **/
/**      CMPR_LICENSE_ACCEPTED=0x80000000      **/
/*****
typedef struct {
    DWORD dwIPAddr;           // client IP address
    DWORD dwStatus;          // client status
    //  UCHAR cLicFlag;       // client license status
} CLIENT_STATUS, *PCLIENT_STATUS;

/*****
/**      All the tGlobalClient related code should be rewritten **/
/**      if the table format is different !!!!!!!!!!!!!!!!!!!!!!! **/
/*****
// can adjust to the real router control table size
#define MAX_TABLEMEMBER 256
CLIENT_STATUS tGlobalClient[MAX_TABLEMEMBER];

// if ctrl-c/ctrl_break
WORD wLoopControl=1;
// WSA flag
BOOL fStarted = FALSE;

BOOL ParseVersion(CHAR *pcVersStr, PCMPVERS pVers)
{

```

```

if (pcVersStr && (*pcVersStr))
{
    INT i = 0;
    CHAR *pStr = pcVersStr;
    CHAR *pDot = strchr(pStr, '.');
    while (pDot)
    {
        *pDot = 0;
        if(*pStr>'9' || *pStr<'0')
            return FALSE;

        pVers->wVers[i] = (WORD) atoi(pStr);
        *pDot = '.';
        pStr = pDot + 1;

        if (++i > 3)
            return FALSE;
        pDot = strchr(pStr, '.');
    }
    if (i==3 && pStr && *pStr<='9' && *pStr>='0')
    {
        pVers->wVers[i] = (WORD) atoi(pStr);
        return TRUE;
    }
}
return FALSE;
}

DWORD StringToAddress(CHAR *pStrAddr)
{
    DWORD dwResult=0;
    DWORD dwTempResult=0;
    if (pStrAddr && (*pStrAddr))
    {
        INT i = 0;
        CHAR *pStr = pStrAddr;
        CHAR *pDot = strchr(pStr, '.');
        while (pDot)
        {
            *pDot = 0;
            if(*pStr>'9' || *pStr<'0')
                return 0;

            if( (dwTempResult=(WORD)atoi(pStr))>255 )
                return 0;

            dwResult|=dwTempResult<<((3-i)*8);
            *pDot = '.';
            pStr = pDot + 1;

            if (++i > 3)
                return 0;
            pDot = strchr(pStr, '.');
        }
        if (i==3 && pStr && *pStr<='9' && *pStr>='0')
        {
            if( (dwTempResult=(WORD)atoi(pStr))<=255 )

```

```

        {
            dwResult|=dwTempResult;
            return dwResult;
        }
    }
}
return 0;
}
// real sample
// need to be rewritten if the client table is different !!!!!!!!!!!
void PrintClientStatus(char* pWhere)
{
    int i, j=0;
    DWORD dwIPAddress,dwTempStatus;
    char sShow[60];
    char* sMessage[3]={"Client Exempt !","Client Timeout !","No Client
response !"};
    char* sMessage2[3]={"Wrong Client Version !","Invalid License !","No
more license !"};
    printf("\n");
    printf("%s\n",pWhere);

    for(i=0; i<MAX_TABLEMEMBER; i++)
    {
        dwIPAddress=SwapDWord(tGlobalClient[i].dwIPAddr);
        dwTempStatus=tGlobalClient[i].dwStatus & CMPR_MASK;
        if(dwTempStatus==CMPR_CLIENT_EXEMPT)
            strcpy(sShow,sMessage[0]);
        else if(dwTempStatus>256)
            sprintf(sShow,"Time Stamp=%d",dwTempStatus-256);
        else if(dwTempStatus>39)
            sprintf(sShow,"Status=%d",dwTempStatus);
        else if(dwTempStatus>32)
            strcpy(sShow,sMessage2[dwTempStatus-33]);
        else if(dwTempStatus>2)
            sprintf(sShow,"Status=%d",dwTempStatus);
        else if(dwTempStatus>0) // shouldn't happen
            strcpy(sShow,sMessage[dwTempStatus]);
        else
            continue;

        printf("<IP address=%d.%d.%d.%d> <License status=%d> <%s>\n",
(dwIPAddress & 0xff000000)>>24,(dwIPAddress & 0x00ff0000)>>16,
(dwIPAddress & 0x0000ff00)>>8, dwIPAddress & 0x000000ff,
tGlobalClient[i].dwStatus & CMPR_LICENSE_ACCEPTED ? 1 : 0,sShow);
        j++;
    }
    if(!j)
        printf("No client information !\n");
}

// real sample
// need to be rewritten if the client table is different !!!!!!!!!!!
void InitializeGlobals(DWORD dwIPAddress)
{
    int i;
    // CLIENT_STATUS initStatus={0,0,0};
    // CLIENT_STATUS initStatus={0,0};
}

```

"00000" "00000" "00000"

```

        dwIPAddress=dwIPAddress & SwapDWord(SUBNET_MASK);
        // assume IPAddr starts from 192.168.0.0
for(i=0; i<MAX_TABLEMEMBER; i++)
{
    memcpy(&tGlobalClient[i],&initStatus,sizeof(CLIENT_STATUS));
    tGlobalClient[i].dwIPAddr=SwapDWord(dwIPAddress);
    if((i%100)==8)
        tGlobalClient[i].dwStatus=CMPR_CLIENT_EXEMPT;
        dwIPAddress++;
}
    PrintClientStatus("=====Initialize=====");
}

/* validate license key, 27 characters
*/
BOOL CMPValidateLicenseKey(char* InputKey)
{
    int i;
    if(strlen(InputKey) != 27)
        return FALSE;
    for(i=0; i<27; i++)
    {
        if(InputKey[i]<'0' ||
            (InputKey[i]>'9' && InputKey[i]<'A') ||
            (InputKey[i]>'Z' && InputKey[i]<'a') ||
            InputKey[i]>'z')
            return FALSE;
    }
    return TRUE;
}

/*
return: DWORD, packet size
1. buffers
    UCHAR* packetChallenge
    UCHAR* packetEncrypted
2. challenge arguments
    DWORD dwRouteIPAddress
    UCHAR* pcmpRouteVersion
    DWORD dwRouteSessionID
    DWORD dwResponseTime
3. user prompt option argument
    char* pUserPrompt
*/
DWORD PrepareUserPromptPacket(UCHAR* packetChallenge, UCHAR* packetEncrypted,
                             DWORD dwRouteIPAddress, UCHAR* pcmpRouteVersion,
                             DWORD dwRouteSessionID, DWORD dwResponseTime, char*
pUserPrompt)
{
    PRT_CHALLENGE_PACKET pTempChallenge;
    WORD wTempPacketSize,wTempOptionStart;
    DWORD dwPresentTime;
    BOOL rStatus;

    // create non heartbeat challenge packet
    dwPresentTime=GetTickCount()/1000;
    rStatus=CMPCreateChallengePacket(packetChallenge, CMP_MAX_CHALLENGE,
CMPP_LINKSYS,

```



```

pTempChallenge=(PRT_CHALLENGE_PACKET)packetChallenge;
wTempOptionStart=pTempChallenge->wPacketSize;
if(rStatus)
    rStatus=CMPPAddLicenseOption(packetChallenge, CMP_MAX_CHALLENGE,
wNumberOfUsers, pLicenseKey);

if(rStatus)
{
    wTempPacketSize=pTempChallenge->wPacketSize;
    // swap
    SwapRouterChallenge((PRT_CHALLENGE_PACKET)packetChallenge);

SwapOptLicense((POPT_ZL_LICENSE)((UCHAR*)packetChallenge+wTempOptionStart));
    // checksum
    pTempChallenge->dPacketCRC=Checksum((WORD*)packetChallenge,
wTempPacketSize, 0);
    // ecrypt the packet
    return CMPEncryptPacket(packetChallenge, wTempPacketSize,
packetEncrypted, CMP_MAX_CHALLENGE);
}
return 0;
}

/*
return: DWORD, packet size
1. buffers
    UCHAR* packetChallenge
    UCHAR* packetEncrypted
2. challenge arguments
    DWORD dwRouteIPAddress
    UCHAR* pcmpRouteVersion
    DWORD dwRouteSessionID
    DWORD dwResponseTime
3. ZAP version option argument
    UCHAR* cmpMinimumVersion
*/
DWORD PrepareVersionInquiryPacket(UCHAR* packetChallenge, UCHAR*
packetEncrypted,
                                DWORD dwRouteIPAddress, UCHAR* pcmpRouteVersion,
DWORD dwRouteSessionID,
                                DWORD dwResponseTime,UCHAR* cmpMinimumVersion)
{
    PRT_CHALLENGE_PACKET pTempChallenge;
    WORD wTempPacketSize,wTempOptionStart;
    DWORD dwPresentTime;
    BOOL rStatus;
    // create non heartbeat challenge packet
    dwPresentTime=GetTickCount()/1000;
    rStatus=CMPPCreateChallengePacket(packetChallenge, CMP_MAX_CHALLENGE,
CMPP_LINKSYS,
                                dwRouteIPAddress, pcmpRouteVersion,
dwRouteSessionID, dwResponseTime, dwPresentTime);
    pTempChallenge=(PRT_CHALLENGE_PACKET)packetChallenge;
    wTempOptionStart=pTempChallenge->wPacketSize;

    if(rStatus)
        rStatus=CMPPAddClientVersionOption(packetChallenge, CMP_MAX_CHALLENGE,
CMPP_ZAPRO, cmpMinimumVersion);

```

```

if(rStatus)
{
    wTempPacketSize=pTempChallenge->wPacketSize;
    // swap
    SwapRouterChallenge((PRT_CHALLENGE_PACKET)packetChallenge);

SwapOptClientVersion((POPT_CLIENT_VERSION)((UCHAR*)packetChallenge+wTempOption
Start));
    // checksum
    pTempChallenge->dPacketCRC=Checksum((WORD*)packetChallenge,
wTempPacketSize, 0);
    // ecrypt the packet
    return CMPEncryptPacket(packetChallenge, wTempPacketSize,
packetEncrypted, CMP_MAX_CHALLENGE);
}
return 0;
}

/*
return: DWORD, packet size
1. buffers
    UCHAR* packetChallenge
    UCHAR* packetEncrypted
2. challenge arguments
    DWORD dwRouteIPAddress
    UCHAR* pcmpRouteVersion
    DWORD dwRouteSessionID
    DWORD dwResponseTime
*/
DWORD PrepareHeartBeatPacket(UCHAR* packetChallenge, UCHAR* packetEncrypted,
    DWORD dwRouteIPAddress, UCHAR* pcmpRouteVersion, DWORD dwRouteSessionID,
    DWORD dwResponseTime)
{
    PRT_CHALLENGE_PACKET pTempChallenge;
    WORD wTempPacketSize;
    DWORD dwPresentTime;
    BOOL rStatus;
    // create non heartbeat challenge packet
    dwPresentTime=GetTickCount()/1000;
    rStatus=CMPCreateChallengePacket(packetChallenge, CMP_MAX_CHALLENGE,
    CMPP_LINKSYS,
        dwRouteIPAddress, pcmpRouteVersion,
    dwRouteSessionID, dwResponseTime, dwPresentTime);
    if(rStatus)
    {
        pTempChallenge=(PRT_CHALLENGE_PACKET)packetChallenge;
        wTempPacketSize=pTempChallenge->wPacketSize;

        SwapRouterChallenge((PRT_CHALLENGE_PACKET)packetChallenge);
        // checksum
        pTempChallenge->dPacketCRC=Checksum((WORD*)packetChallenge,
wTempPacketSize, 0);
        // ecrypt the packet
        return CMPEncryptPacket(packetChallenge, wTempPacketSize,
packetEncrypted, CMP_MAX_CHALLENGE);
    }
    return 0;
}

```

```

/*
return: int, socket status
Windows special
*/
int CMPGetSocket(SOCKET *pSock, DWORD dwIPAddr, INT iTimeout)
{
    INT iStatus = SOCKET_ERROR;
    SOCKET sock = INVALID_SOCKET;
    WSADATA wsaData;

    int iRetVal;
    iRetVal = WSASStartup ( MAKEWORD ( 1,1 ), &wsaData );
    if (iRetVal != 0)
    {
        printf( "/n WSASStartup=%d", iRetVal );
        return iStatus;
    }

    fStarted = TRUE;
    sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if (sock == INVALID_SOCKET)
        iStatus = SOCKET_ERROR;
    else if (dwIPAddr)
    {
        struct sockaddr_in addr;
        memset(&addr, 0, sizeof(addr));
        addr.sin_family = AF_INET;
        addr.sin_port = SwapWord((WORD)CMP_PORT);
        addr.sin_addr.s_addr = dwIPAddr;
        iStatus = bind(sock, (const struct sockaddr *) &addr,
sizeof(addr));
        if (iStatus)
        {
            printf( "\n bind=%d", WSAGetLastError ( ) );
            closesocket(sock);
            sock = INVALID_SOCKET;
        }
    }

    // do we ever want to set a send timeout?
    if (iTimeout && !iStatus)
        iStatus=setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, (char *)
&iTimeout, sizeof(iTimeout));

    if (iStatus)
    {
        printf( "\n setsockopt=%d", WSAGetLastError ( ) );
    }
    *pSock = sock;
    return iStatus;
}

/*
return: DWORD, length received from the client
Windows special
*/
DWORD CMPListenToClient(SOCKET rsock, DWORD dwServerAddr,

```

```

                                UCHAR* cBufReceived, DWORD dwBufLen, DWORD*
dwClientIP)
{
    INT iLen=0;
    int iSize;
    struct sockaddr_in remAddr;

    memset(cBufReceived, 0, dwBufLen);
    memset(&remAddr, 0, sizeof(remAddr));
    iSize = sizeof(remAddr);
    if(rsock!=INVALID_SOCKET)
        iLen = recvfrom(rsock, cBufReceived, dwBufLen, 0, (struct sockaddr
*) &remAddr, &iSize);
    if (iLen > 0)
    {
        if (((remAddr.sin_addr.s_addr & SUBNET_MASK) == (dwServerAddr &
SUBNET_MASK)) &&
            ((remAddr.sin_port == SwapWord((WORD)CMP_PORT))))
        {
            *dwClientIP=remAddr.sin_addr.s_addr;
            return iLen;
        }
    }
    return 0;
}

/* close socket
Windows special
*/
void CMPKillSocket(SOCKET rSock)
{
    if (rSock != INVALID_SOCKET)
        closesocket(rSock);
    if ( TRUE == fStarted )
    {
        WSACleanup ( );
    }
    fStarted=FALSE;
}

//
// Broadcasts challenge packet
// Windows special
void CMPBroadcast(SOCKET rSock, UCHAR* packetChallenge, DWORD dwPkLen, DWORD
dwServerAddr)
{
    int iSendOut;
    struct sockaddr_in remAddr;
    memset(&remAddr, 0, sizeof(remAddr));
    remAddr.sin_family = AF_INET;
    remAddr.sin_port = SwapWord((WORD)CMP_PORT);
    remAddr.sin_addr.s_addr = SUBNET_BROADCAST_ADDR(dwServerAddr,
SUBNET_MASK);
    if(rSock!=INVALID_SOCKET)
        iSendOut=sendto(rSock, (const char *)packetChallenge, dwPkLen, 0,
(const struct sockaddr *) &remAddr, sizeof(remAddr));
}

```


to talk to cliens with a license inquiry and update the client table, rtest /l
license string

to authorize traffic, rtest /t IP address

```

*/
void main(int argc, char** argv)
{
    BOOL bSettings=FALSE;
    //input from the control panel
    //ZAP related
    BOOL bZAPEnabled=TRUE;
    DWORD gdwLinkSysZoneLabsWeb=0;
    DWORD gdwResponseTime=3; // client repsonce time
    char* newUserPrompt="LinkSys security enforcement"; //
    CMPVERS gcmpZAPVersion;
//    "5-5-5-6-6" "AQ13E-B2SGE-93JSE-HGXY36-YS6TT8"; //27 CHARACTERS ?
    char* newLicenseKey="jsgn7rffsigik024sub23drjp00"; //27 CHARACTERS ?
    char cUserLicense[28];
    char cUserPrompt[60];

    // router info
    CMPVERS gcmpRouteVersion;
    DWORD gdwServerAddr; //192.168.0.1
    DWORD gdwRouteSessionID=0;

    // return from client and to be saved in client table
    DWORD gdwclientStatus, gdwClientIP;
    // when authorize connection
    // redirect to sandbox
    DWORD gdwDstIPAddr=0;
    WORD gwPortDst=80;
    // number of licences being assigned
    WORD gwNumberOfUsers=0;

    int iIndex;
    DWORD atReturn;
    // socket, subnet address
    SOCKET socketCMP;
    int iTimeout=5000; //receive time out
    //packet space
    char packetInOut[CMP_MAX_CHALLENGE], packetRSA[CMP_MAX_CHALLENGE];

    DWORD tcStart;
    DWORD tcNow;
    DWORD dwPresentTime;

    BOOL rStatus;
    DWORD dwLenReturned;
    // hard code ZAP version
    gcmpZAPVersion.wVers[0]=2;
    gcmpZAPVersion.wVers[1]=6;
    gcmpZAPVersion.wVers[2]=0;
    gcmpZAPVersion.wVers[3]=88;

    // hard code router version
    gcmpRouteVersion.wVers[0]=2;
    gcmpRouteVersion.wVers[1]=6;
    gcmpRouteVersion.wVers[2]=8;

```

"5-5-5-6-6"

```

gcmpRouteVersion.wVers[3]=6;
// clean buffer
memset(packetInOut,0,CMP_MAX_CHALLENGE);
memset(packetRSA,0,CMP_MAX_CHALLENGE);
// set event for exiting loop
if(FALSE==SetConsoleCtrlHandler((HANDLER_ROUTINE)CtrlHandler,TRUE))
    return;

gdwServerAddr=0xc0a80001;//192.168.0.1
// Initialize client table
InitializeGlobals(gdwServerAddr);
// swap IP address
gdwServerAddr=SwapDWord(gdwServerAddr);
if(CMPGetSocket(&socketCMP, gdwServerAddr, iTimeout)==SOCKET_ERROR)
    return;
// check command line arguments
if ( argc>1 && (( *argv[1] == '-' ) || ( *argv[1] == '/' ) ) )
{
    //talk with client through challenge/hello/response
    if(*(argv[1]+1)=='c' || *(argv[1]+1)=='C' ||
        *(argv[1]+1)=='t' || *(argv[1]+1)=='T' ||
        *(argv[1]+1)=='a' || *(argv[1]+1)=='A' ||
        *(argv[1]+1)=='l' || *(argv[1]+1)=='L' ||
        *(argv[1]+1)=='v' || *(argv[1]+1)=='V' ||
        *(argv[1]+1)=='s' || *(argv[1]+1)=='S' ||
        *(argv[1]+1)=='p' || *(argv[1]+1)=='P' )
    {
        dwLenReturned=0;
        // first packet is a control panel setting packet
        //talk with client through challenge/hello/response
        //!!!!!!!!!!!!!!
        // the change should be reflected in every heartbeat
        // please pick up code for different combinations
        //!!!!!!!!!!!!!!
        // ***** challenge + any
option starts
        if(*(argv[1]+1)=='s' || *(argv[1]+1)=='S')//send control
panel setting, not tested
        {
            // ZAP related
            //new check time, response time=check time/2
            //new download site, default linksys.zonelabs.com
            //new user prompt
            //new required client version
            //new license key
            WORD wTempPacketSize;
            PRT_CHALLENGE_PACKET pTempChallenge;
            bSettings=FALSE;
            bSettings=CMPCreateChallengePacket(packetInOut,
CMP_MAX_CHALLENGE, CMPP_LINKSYS, gdwServerAddr,
                (UCHAR*)&gcmpRouteVersion,
gdwRouteSessionID, gdwResponseTime, GetTickCount()/1000);
            pTempChallenge=(PRT_CHALLENGE_PACKET)packetInOut;
            //ZAP related
            if(bZAPEnabled)
            {
/*****
// UserPrompt sample code begins

```

```

#ifdef DO_USERPROMPT
                                // user prompt packet if changed
                                if(bSettings)
                                {

wTempPacketSize=pTempChallenge->wPacketSize;

bSettings=CMPPAddUserPromptOption(packetInOut, CMP_MAX_CHALLENGE,
newUserPrompt);

SwapOptUserPrompt((POPT_USER_PROMPT)((UCHAR*)packetInOut+wTempPacketSize));
                                }
#endif
// UserPrompt sample code ends
/*****
                                // licence issued
                                if(bSettings)
                                {

wTempPacketSize=pTempChallenge->wPacketSize;
                                bSettings=CMPPAddLicenseOption(packetInOut,
CMP_MAX_CHALLENGE, gwNumberOfUsers, newLicenseKey);

SwapOptLicense((POPT_ZL_LICENSE)((UCHAR*)packetInOut+wTempPacketSize));
                                }
                                // version packet if changed
                                if(bSettings)
                                {

wTempPacketSize=pTempChallenge->wPacketSize;

bSettings=CMPPAddClientVersionOption(packetInOut, CMP_MAX_CHALLENGE,
CMPP_ZAPRO, (UCHAR*)&gcmpZAPVersion);
                                SwapOptClientVersion((POPT_CLIENT_VERSION)
((UCHAR*)packetInOut+wTempPacketSize));
                                }
                                // checksum and encrypt
                                if(bSettings)
                                {
                                        wTempPacketSize=pTempChallenge->wPacketSize;

SwapRouterChallenge((PRT_CHALLENGE_PACKET)packetInOut);
                                        // checksum

pTempChallenge->dPacketCRC=Checksum((WORD*)packetInOut, wTempPacketSize, 0);
                                        // ecrypt the packet
                                        dwLenReturned=CMPEncryptPacket(packetInOut,
wTempPacketSize, packetRSA, CMP_MAX_CHALLENGE);
                                }

} //!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
// ***** challenge + any
option ends

/*****
// UserPrompt sample code begins
#ifdef DO_USERPROMPT

```

```

// first packet is a user prompt packet
//talk with client through challenge/hello/response
if(*(argv[1]+1)=='p' || *(argv[1]+1)=='P')
{
    if(argc>2)
    {
        if(strlen(argv[2])<60)
            strcpy(cUserPrompt,argv[2]);
    }
    else
        strcpy(cUserPrompt,newUserPrompt);
    dwLenReturned=PrepareUserPromptPacket(packetInOut,
packetRSA,
                                gdwServerAddr,
(CHAR*)&gcmpRouteVersion, gdwRouteSessionID,
                                gdwResponseTime, cUserPrompt);
}
#endif
// UserPrompt sample code ends
/*****/
// first packet is aversion packet
//talk with client through challenge/hello/response
if(*(argv[1]+1)=='v' || *(argv[1]+1)=='V')
{
    if(argc>2)
    {
        //201
        if(ParseVersion(argv[2], &gcmpZAPVersion)==FALSE)
        {
            gcmpZAPVersion.wVers[0]=2;
            gcmpZAPVersion.wVers[1]=7;
            gcmpZAPVersion.wVers[2]=0;
            gcmpZAPVersion.wVers[3]=0;
        }
    }
    dwLenReturned=PrepareVersionInquiryPacket(packetInOut,
packetRSA,
                                gdwServerAddr,
(CHAR*)&gcmpRouteVersion, gdwRouteSessionID,
gdwResponseTime, (CHAR*)&gcmpZAPVersion);
}

// first packet is license key packet
//talk with client through challenge/hello/response
if(*(argv[1]+1)=='l' || *(argv[1]+1)=='L')
{
    strcpy(cUserLicense,newLicenseKey);
    if(argc>2)
    {
        //201
        if(strlen(argv[2])==27 &&
CMPValidateLicenseKey(argv[2]))
            strcpy(cUserLicense,argv[2]);
        else
            strcpy(cUserLicense,newLicenseKey);
    }
    dwLenReturned=PrepareLicenseKeyPacket(packetInOut,
packetRSA,

```

T09020"ES9E0E09

```

                                gdwServerAddr,
(CHAR*)&gcmpRouteVersion, gdwRouteSessionID,
                                gdwResponseTime, gwNumberOfUsers,
cUserLicense);
    }

    //talk with client through challenge/hello/response
    if(*(argv[1]+1)=='c' || *(argv[1]+1)=='C' ||
*(argv[1]+1)=='t' || *(argv[1]+1)=='T')
        dwLenReturned=PrepareHeartBeatPacket(packetInOut,
packetRSA,
                                gdwServerAddr, (CHAR*)&gcmpRouteVersion,
gdwRouteSessionID, gdwResponseTime);
    if(dwLenReturned)
        CMPBroadcast(socketCMP, packetRSA, dwLenReturned,
gdwServerAddr);
    // to get the current tick to control the heartbeat
frequency
    // ***** main
loop starts
    // ***** main
loop starts

tcStart = GetTickCount();
while (wLoopControl)
{
    // ***** periodical listen and analysis
starts
        if(dwLenReturned=CMPListenToClient(socketCMP,
gdwServerAddr, packetInOut, CMP_MAX_CHALLENGE, &gdwClientIP))
        {
            // after receiving a packet successfully
            if(dwLenReturned=CMPDecryptPacket(packetInOut,
dwLenReturned, packetRSA, CMP_MAX_CHALLENGE))
            {

if(rStatus=CMPSanalyseDecryptedPacket(packetRSA, dwLenReturned,
&gdwclientStatus))
                {
                    if(rStatus==CMPM_CLIENT_HELLO)
                    {
                        // send response to heart beat
immediately with response time to 0

dwLenReturned=PrepareHeartBeatPacket(packetInOut, packetRSA,
                                gdwServerAddr,
(CHAR*)&gcmpRouteVersion, gdwRouteSessionID, 0);
                                if(dwLenReturned)
                                    CMPBroadcast(socketCMP,
packetRSA, dwLenReturned, gdwClientIP);
                                }
                            else
if(rStatus==CMPM_CLIENT_RESPONSE)//update sample client table
                            {

iIndex=SwapDWord(gdwClientIP)-SwapDWord(tGlobalClient[0].dwIPAddr);
                                //the status includes CMP and license status
                                /*****/

```



```

// ***** demo of checking traffic
periodically starts
tested
{
    char stringIP[28];
    gdwDstIPAddr=SwapDWord(gdwClientIP);
    sprintf(stringIP,"%d.%d.%d.%d", (gdwDstIPAddr &
0xff000000)>>24, (gdwDstIPAddr & 0x00ff0000)>>16,
    (gdwDstIPAddr & 0x0000ff00)>>8, gdwDstIPAddr
& 0x000000ff);

    if( (*(argv[1]+1)=='t' || *(argv[1]+1)=='T'))
    {
        gdwDstIPAddr=0;
        if(argc>2)
        {
            // string to IP address

gdwDstIPAddr=StringToAddress(argv[2]);
            if(gdwDstIPAddr==0)
                printf("\n Wrong IP address
<%s> inputed !! sample <192.66.55.218>",argv[2]);
            else
                strcpy(stringIP,argv[2]);
        }
        else
            printf("\n Please input IP
address,format: router /t 192.66.55.218");
    }

    if(gdwDstIPAddr)
    {
        // when to connect to the internet
        dwPresentTime=GetTickCount()/1000;
        // presume IP address is in a continuous
range starting from 192.168.0.0

iIndex=gdwDstIPAddr-SwapDWord(tGlobalClient[0].dwIPAddr);
        if(iIndex<0 || iIndex>255)
            printf("\n Not router related IP
address <%s> inputed !!",stringIP);
        else
        {
            atReturn=CMPTAuthorizeTraffic(gwPortDst, tGlobalClient[iIndex].dwStatus &
CMPR_MASK, dwPresentTime, gdwResponseTime*4);
            if(atReturn==0)
                printf("\nThe traffic request
by %s is authorized to pass !",stringIP); //pass
            else if(atReturn>0)//redirect to
LinkSys.ZoneLabs.com, port number = atReturn
                printf("\nThe traffic request
by %s is directed to Linksys.ZoneLabs.com:%d",stringIP,atReturn);
            else
                printf("\nThe traffic request
by %s is blocked !",stringIP); //block
        }
    }
}

```

FOR9020-ES9E0E0E9

```

    }
    wLoopControl++;
    // ***** demo of checking traffic
periodically ends
    }
loop ends // ***** main
loop ends // ***** main
    }
}
else
{
    printf("\n *To end the heartbeat, type CTRL-C/CTRL-BREAK*");
    printf("\n ***** To test heartbeat *****");
    printf("\n router /p prompt string");
    printf("\n example: router /p ZAP is enforced to run");
    printf("\n router /v version number");
    printf("\n example : router /v 2.6.0.300");
    printf("\n router /l license string");
    printf("\n example: router /l abcdefghijklmnopqrstuvwxyz1");
    printf("\n router /s");
    printf("\n default settings plus heartbeat");
    printf("\n router /c");
    printf("\n heartbeat");
    printf("\n router /a");
    printf("\n example: not available");
    printf("\n ***** To test authorizing traffic *****");
    printf("\n router /t IP address");
    printf("\n example: router /t 192.66.55.218");
}
CMPKillSocket(socketCMP);
}

```

T03040" E03040