The Capture-HPC client architecture

Radek Hes, Victoria University of Wellington
Ramon Steenson, Victoria University of Wellington,
Christian Seifert, Microsoft Inc.

NZ chapter, Capture-HPC alliance.
Second revision August 2010

Abstract

This paper documents the architecture of the Capture-HPC client. It is hoped that using this document, developers will form a clearer understanding of the implementation of Capture-HPC and use this as a base from which to explore areas of research within high interaction client honey-pots and extend functionality of the HPC client. This document supports the documentation of the honey client code and should not be used as a replacement for the code.

A brief introduction to the Capture-HPC system is provided before describing the entities and relationships of the objects in the client. Event traces are presented which describes the communication flow given the set of possible system events.

1. Introduction, what is the Capture-HPC?

The Capture-HPC is a high interaction Honeypot; a security application which actively searches for malicious servers that attack web browsers or any other application (such acrobat reader) which access remote material or the underlying operating system. This differs from the low interactive form of the conventional passive-honeypot system that simply awaits attacks. The client presents itself to the potentially malicious non-local server as an authentic operating system and application set which; unknown to the attacker site exists within a VMWare-based virtual environment enriched with logging and analysis tools which detect malicious activity. This includes network loggers, kernel/registry loggers, file creation and deletion event loggers amongst many other possibilities. The environment may be distributed amongst many hosts or single hosts with many VM- environments which add to the scalability of the search technique.

1.1 HoneyNet System operation overview

The Capture-HPC system consists of both client and server subsystem. In typical operation the server is run on one host machine and one or more capture clients are run within their own VMWare virtual environments, on the same or independent physical hosts.

The server initially connects to the virtual machine via the VMWare VIX library over the network and invokes the Capture client within the virtual environment. The capture client initialises a set of system monitors including file system, registry, process and network monitors and forms a socket communication channel for exchange of information between client and server.
Once a communication channel is established the server issues a batch of visit events to the capture client which in turn directs a set of web-browsers or applications to visit the website or remote material. Currently supported applications include Safari, Firefox and Internet Explorer web browsers, Adobe reader, Open office writer and Microsoft office applications.
During the visitation the Capture client monitors the virtual machine state for any changes to the underlying operating environment and if any changes are found the capture client logs the event and sends the information back to the server. At the occurrence of any malicious events the capture server also logs the newly received information, resets the virtual machine to a base state and continues with further visitations.

Figure 1 below provides an overview of the client components and Figure 2 provides an overview of the CaptureHPC system.

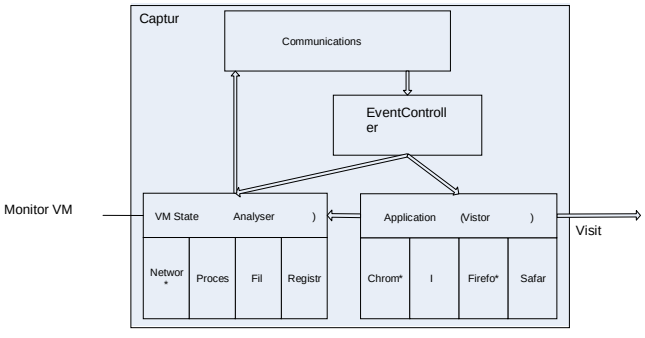

Figure 1 Client components.

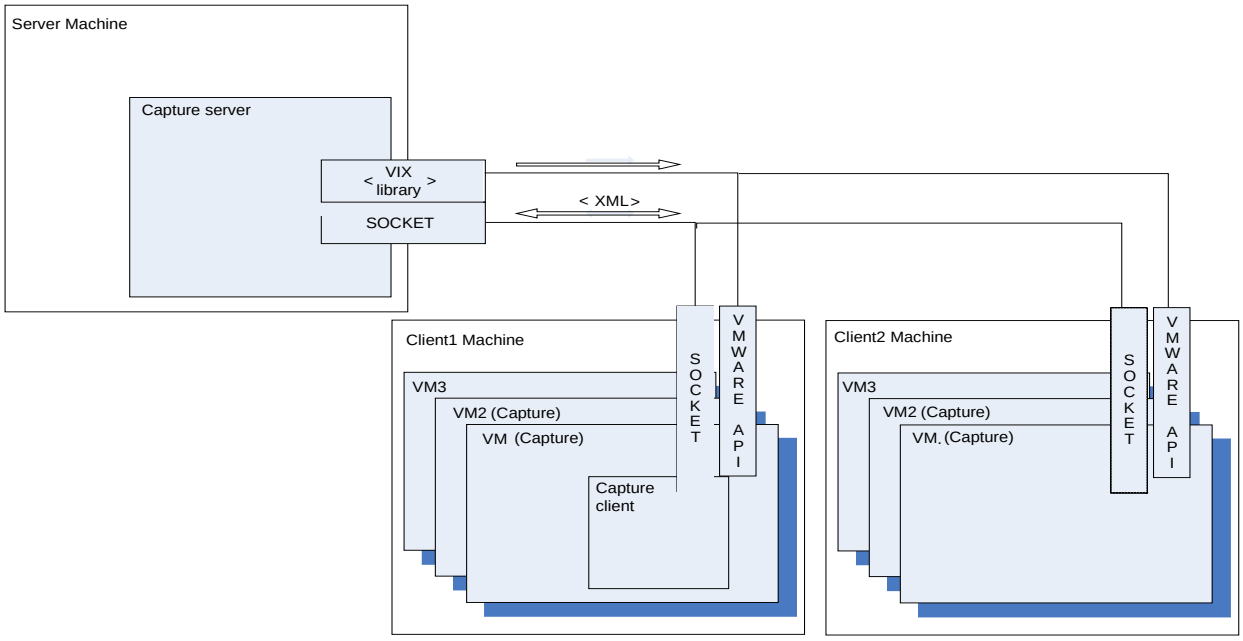

Figure 2 Capture-HPC system overview.

2. Client architecture

The following section describes the object and object interactions within the client. The architecture of the server is not approached in this document.

2.1 The Client

The client itself is composed of a set of four subsystems; the Client/Server communication, Visitor, Analyzer and Event-Controller (internal communications) as shown in figure 1.

2.2 Communication subsystem

The communications subsystem provides a protocol for communication between the server entity and the local client. This subsystem is composed of two objects: a Transmitter -<Server> and Receiver <ServerReceive>. Messages are received from the remote server by the client side Receiver. The Receiver stub runs in its own execution thread and simply polls on XML based server messages -either as Pings or <VisitEvent> notifications.

Once a message is received the receiver passes the XML messages <EventManager>, which then decrypts the XML and signals object functions such as starting and stopping monitors and application plug-ins.

2.3 The Visitor subsystem

The Visitor subsystem is responsible for directing applications (including web-browsers) to the URL's, sent to the Capture-HPC client for processing. Upon an initialisation call from the server (via the <EventMonitor>) the <Visitor> initialises all of the Visitor-plug-ins including IE plug-in, Safari, Adobe reader, Firefox, Opera, Microsoft Word, and OO writer.

These plug-ins all implement a common interface the <<ApplicationPlugin>> which includes the method <VisitGroup(VisitEvent)> -which visits a list of URL's. The Visitor then connects to the <EventManager> in order to receive <VisitEvents> issued by the Capture server.

The <Visitor> object operates in a separate thread when executing each plug-in and will block inside the plug-in while it waits for the <VisitEvent> to complete.

The <Visitor> keeps the <Analyzer> well informed along every step of the visitation process; prior to the <ApplicationPlugin> visit event, upon completion of the <ApplicationPlugin> and at occurrence of any errors during the <VisitEvent>. The <Analyzer> reports these events via the communications subsystem to the remote server.

Visitation of applications may be run in three different modes, sequential, bulk and divide and conquer visitation. Sequential mode performs visitation of the URL set in a sequential fashion. The bulk mode performs a set of visit requests all at once. The DAC method involves the applications visiting the websites in bulk mode and upon a malicious event detected will perform a divide and conquer algorithm to isolate the offending site.

2.4 The Analyzer subsystem

The Analyzer subsystem is responsible for maintaining the various virtual machine monitors within the system. Presently the monitors include a file system monitor, process monitor, registry monitor and the network monitor. The file system monitor detects creation, opening, reading, writing and deletion of files. The process monitor detects any events that the underlying operating system provides as instrumentation. The registry monitors the system registry for all but not limited to the following operations; create, open, delete, delete value, set value, enumerate keys, enumerate value,

query, query value, and close. The network monitor currently detects opening and closing of ports on any installed interface.

The monitors each run in separate threads, and either poll on kernel events (in the case of the file system) or system level events are triggered by the kernel –in the case of network port monitoring, registry monitoring and process monitors where low lifetimes of events may fairly be expected.

The monitors implement a common interface which has methods start(), stop(), InstallKernelDriver() and LoadExclusionList() / AddExclusion(). Exclusions are a set that is bypassed by the monitor components that would otherwise be erroneously detected by the system monitor as instigated by a malicious site.

Upon instantiation each monitor loads the monitor's kernel driver, creates an exclusion list based on a configuration file, initialises a log file for kernel driver events, and connects itself to the <EventController>, which signals any arrival of new exclusion descriptors straight to the monitors.
Upon the detection of an event a monitor will decode the driver message, check the exclusion list for the event received and finally signal the arrival of the event to the Analyzer.

The Analyser receiving this event will package any event details and send the event package to the communication subsystem to relay the messages.

2.5 The Event-Controller (and internal communications)

All communications from the server and between the monitor objects and visitor objects are coordinated by the <EventController>. The event controller is responsible from decoding visit requests from the server, initiating the visitor object –which in turn controls the visitation process, and initiates the analyser.

All subsystems communicate through signals from the boost STL –private object methods that are invoked by a second privileged object

2.6 Support objects

The <Logger>, <OptionsManager> and <Process managers> are considered support objects. The Logger object logs events to a method of operation.
The <OptionsManager> maintains the globally accessible options within the system. Objects will query the <OptionsManager> to direct execution flow based on the users options provided in command line arguments. Enabling disabling monitors are examples of options available to the system.
The <ProcessManager> is used by the process monitor to map process ID of newly created processes to process paths. This is signalled on a new process event.

3. Event traces, threads and entity relationships.

The following figures provide a pictorial view of the entities of the client honeypot, entity relationships, the threaded objects, and the event traces. It is hoped that the following may be used as a concise operational view of the capture code.
The ownership relationship between entities is shown below by the red solid diamond with the tail connected to the owner object, whereas a dotted red line indicates that the object rooted at the tail maintains a pointer to the object.
Communication lines are drawn in black with multiline bus drawn with crossing lines. Input methods are represented by blocks on the left of the object whilst output methods are written on the right.
Primitive state variables are located at the top right of objects and maintained data structure are on the left.
Event traces are labelled in the order of occurrences and highlighted on the figures.

3.1 Client initialisation event

The Capture client is invoked through the VMWare API by the server object. The capture client instantiates the system then forms a socket communication channel with the Server. The client side <ServerReceive> object then polls on a message from the server object.

3.2 Visit URL event

When a visit event is received by the <ServerReceive> object, it passes the message to the <EventController> which extracts the event type and then signals the <Visitor>::OnServerEvent() method. The OnServerEvent() routine creates a <VisitEvent> object and initialises it with a URL to visit.

3.3 Malicious event

When a monitor object detects a malicious event the monitor calls On(Monitor)Event within the <Analyser> object. The <Analyser> sends the event message to the to the client side Server object. The Server then sends the raw data to the remote server. Figure 6 describes the malicious event process.

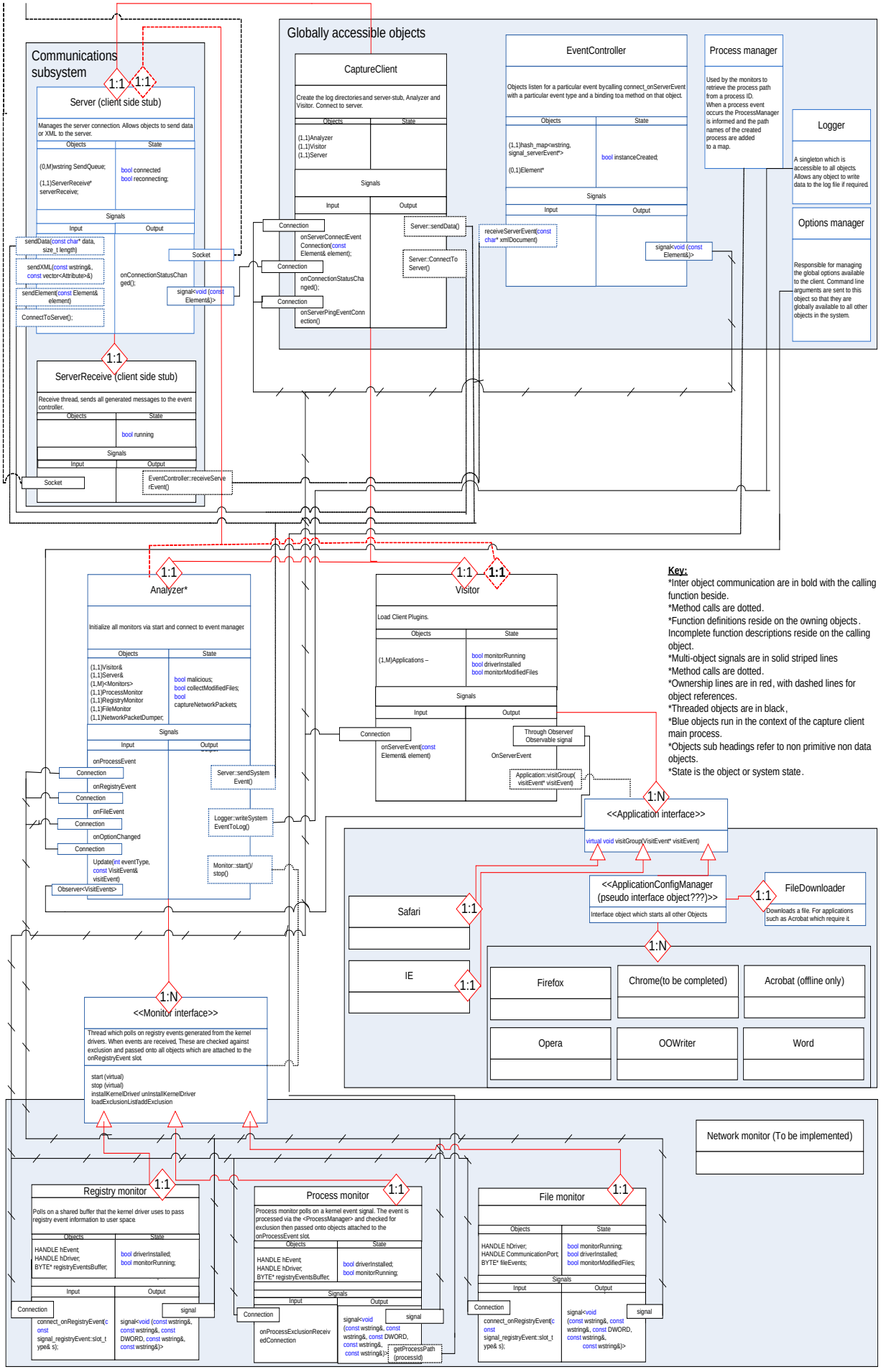

Figure 1, The honey client architecture.

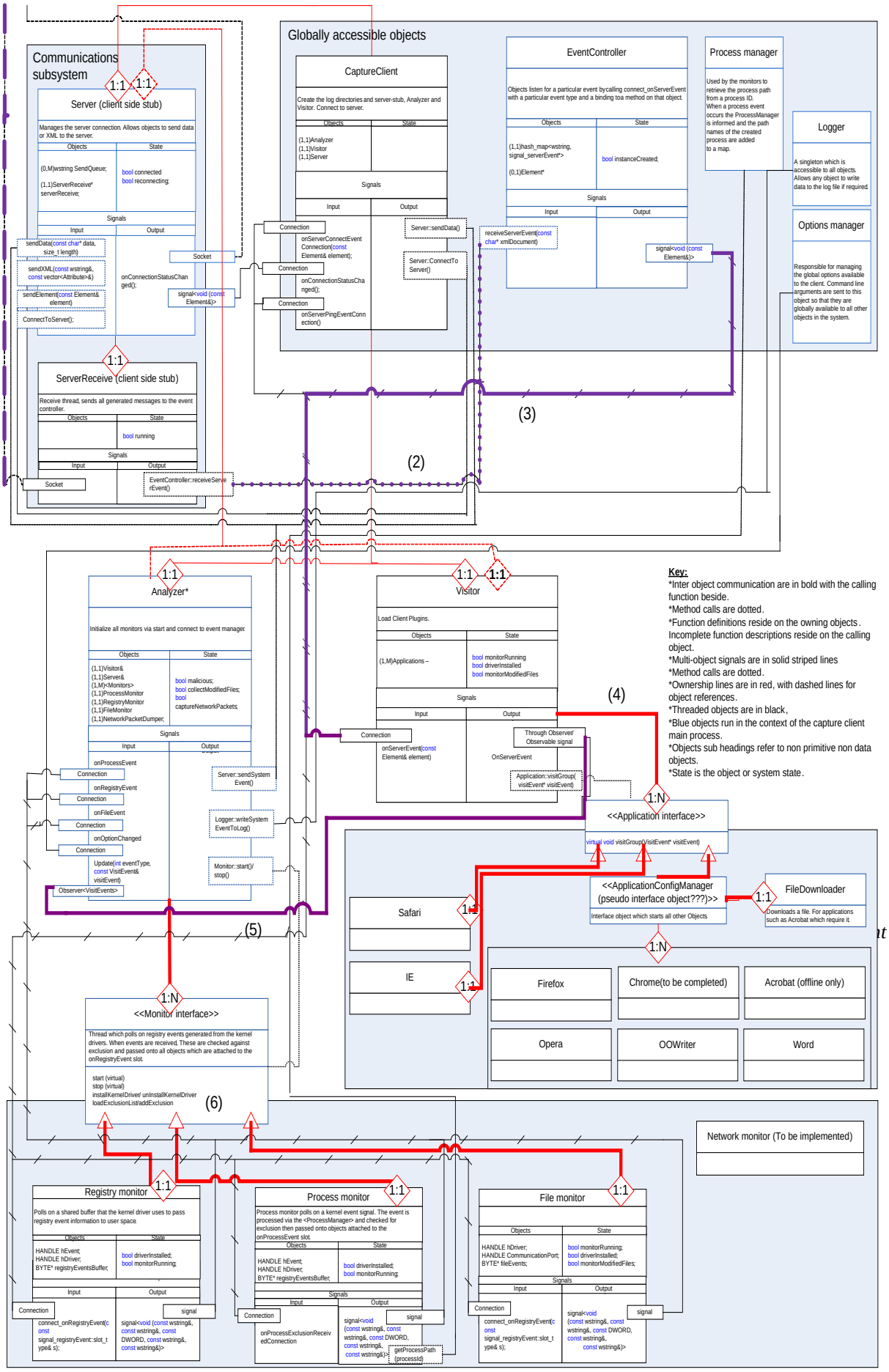

Figure 2,Client Initialisation .

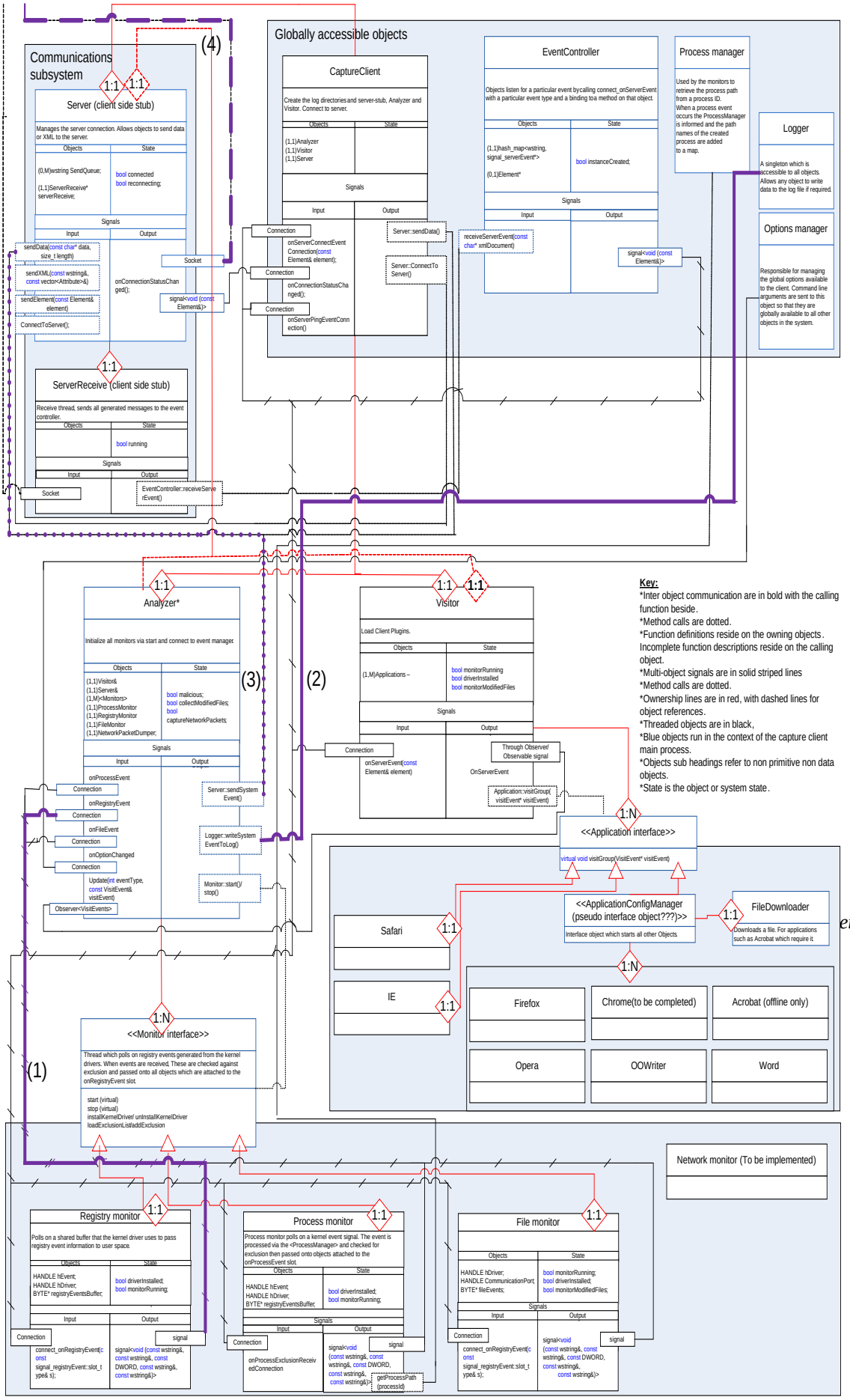

Figure 3,Visit event received from server.

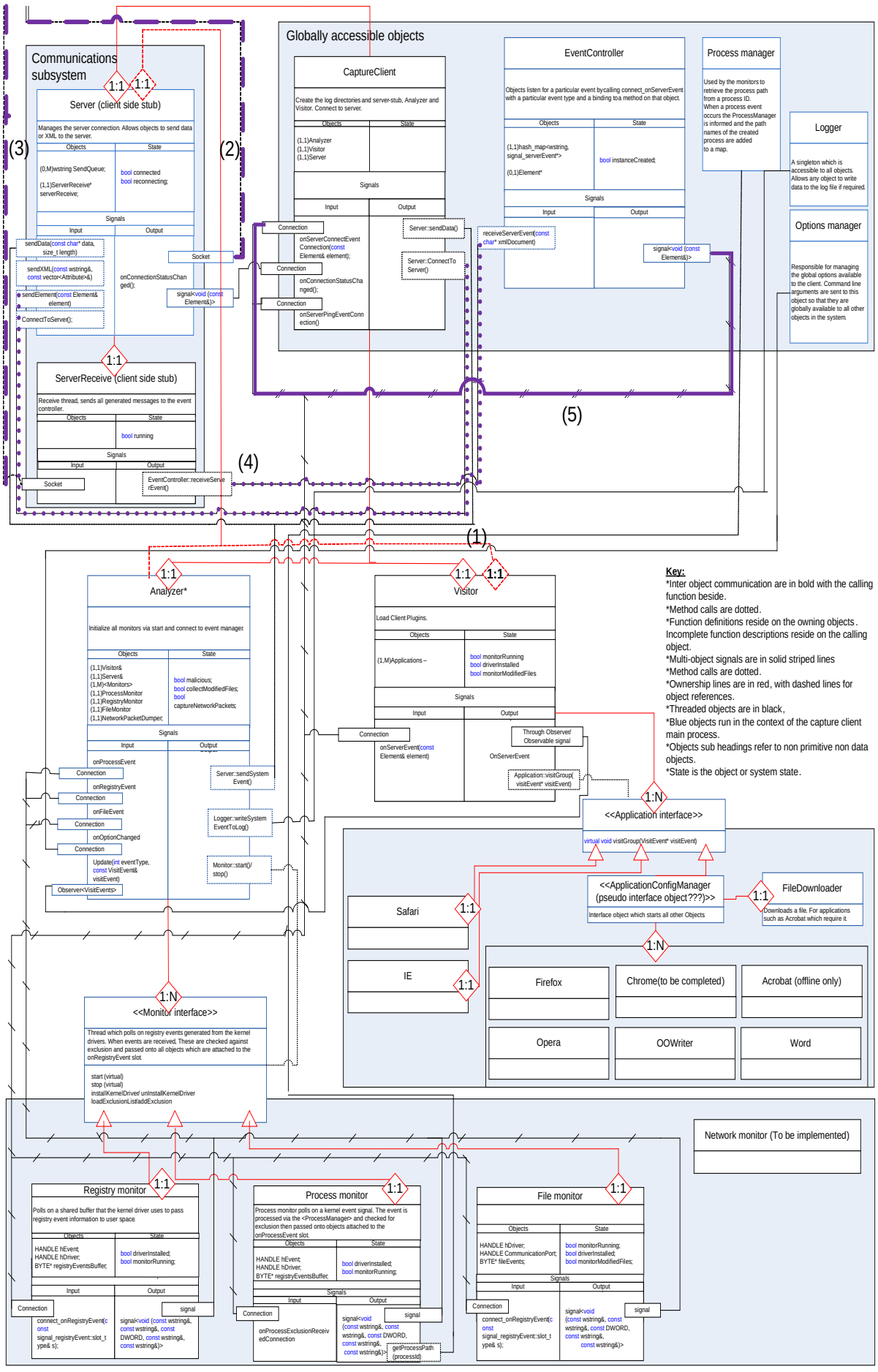

Figure 4,Malicious event detected.

Future extensions for developers

Presently the only supported environment which has software support is Windows XP, however Vista and Windows 7 remain feasible though untested options with the current system monitors. A Unix/linux and Mac variation may also be considered with all but monitor code transportable with limited patching.

Chrome, Flash, IE7 and 8 all to be implemented as potential exploited programs.

References

[1] Taxonomy of Honeypots, Technical Report CS-TR-06/12, Christian Seifert, Ian Welch, Peter Komisarczuk. June 2006.

[2] Capture – A behavioural analysis tool for applications and documents. Christian Seifert, Ian Welch, Peter Komisarczuk, Ramon Steenson, Barbara Endicott-Popovsky. June 2006.

[3] Application of divide-and-conquer algorithm paradigm to improve the detection speed of high interaction client honeypots, Christian Seifert, Ian Welch, Peter Komisarczuk. March 16 2008.

[4] Know your Enemy: Malicious Web Servers. Christian Seifert, Ramon Steenson, Ian Welch, Peter Komisarczuk, Thorsten Holz, Bing Yuan, Michael A. Davis. August 2007.

[5] Improving Detection Speed and Accuracy with Hybrid Client Honeypots, PhD proposal VUW. Christian Seifert.

[6] Honeynet PPT presentation. Available at www.honeynet.org/speaking/honeynet_project-3.0.1.ppt.zip.

[7] BOOST C library. www.boost.org/doc/html/signals.html.

[8] VMWARE API. Available at www.vmware.com/pdf/Prog_API_Prog_Guide.pdf

[9] Capture-HPC-system-guide.pdf, Stirling D.