Automated state machines applied in client honeypots

Yaser Alosefer, Omer Rana
School of Computer Science & Informatics,
Cardiff University,
UK
{Y.Alosefer, O.F.Rana }@cs.cf.ac.uk

Abstract—**Client honeypots visit and interact with suspect web sites in order to detect and collect information about malware. This paper will show the benefits of client honeypots as well as the use of an automated state machine in conjunction with a client honeypot. The state machine provides a powerful framework to organize monitoring of malware activity and record the results. We are developing a platform for integrating client honeypots applied on state machines in a Windows environment.**

Keywords-component; honeypot, client honeypot, malware analysis, automated state machine, web-based malware

I. INTRODUCTION

A Honeypot is defined as a "security resource whose value lies in being probed, attacked or compromised" [1]. The aim of the honeypot is to collect high value data on attacks and attackers by monitoring the state of a real operating system or services. Alternatively it can emulate them to detect traffic that could come to the system. Because the Honeypot is a provision made by the owner of the network, it should not receive any internet traffic from the network or the outside world. Consequently, any traffic that comes to the honeypot could be an attempt to attack or scan the system. Generally honeypots are divided into two main types: passive honeypots and active honeypots (client honeypots).

The main difference between a honeypot and other security techniques such as firewalls or IDS is its log files. The log files of honeypot traffic have a high value for the security team and owner of the network because they reveal the traffic of the attacker without any false positives that could be logged from a firewall or an IDS. By using the honeypot technique, we can avoid the effort required to analyze the log files from other security techniques.

II. CLIENT HONEYPOTS

Attackers these days target client applications such as web browsers or media players whereas they used to threaten servers. Thus we need ways to determine methods of attack and to capture malicious scripts and tools to improve upon security. The active honeypot is a new concept quite different from the passive honeypot. Instead of building a honeypot and waiting passively for attackers, the active honeypot goes out and searches for the attackers. Active honeypots are also called client honeypots or honeyclients. The client honeypot interacts with a server to study it and determine if an attack has happened. The client honeypot can interact with websites over the HTTP service, or with other protocols such as email and ftp.

The client honeypots are divided into two main types based on their level of interaction: low-interaction client honeypots and high-interaction client honeypot. A low-interaction client Honeypot simulates a client used for interacting with a server using a lightweight simulation instead of a real system. The aim of this type is to discover malicious code present within a server by interacting with it and then examining the reply to detect malicious code within the page. On the other hand, a high-interaction client honeypot is a more advanced type that uses a real operating system to examine malicious servers.

The basic concept behind this type is that the operating system itself acts as a client browsing a server that has been identified previously by a queuer just like a human browsing the website. After a visit to the website has been made by the client, the analysis engine checks for evidence of malicious code. There are several aspects of the state of the system to monitor after a website visit: such as Processes, Windows Registry, The File system and ports. Fig. 1 shows the main difference between the high- and low-interaction client honeypots.

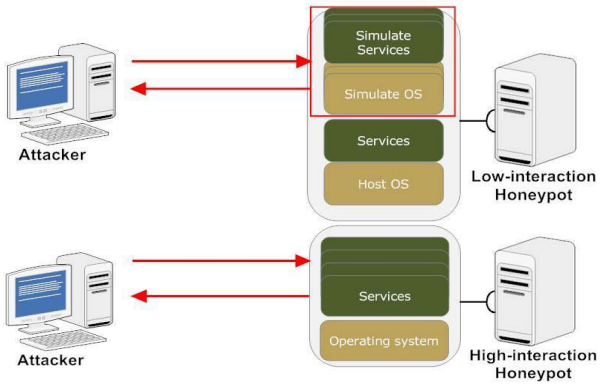

Fig. 1: Architecture of low- and high-interaction client honeypots.

III. STATE MACHINES

In simple terms, a state machine is a machine indicating different states of a task being performed. If we can plan out a

978-1-4244-6949-9/10/$26.00 ©2010 IEEE

strategy in advance, in such a manner that each state gets defined in terms of time and resources, then the probability of completing the task in time increases i.e. the workflows become much smooth. A workflow defined as "a series of steps, decisions, and rules needed to complete a specific task" [2]. There are essentially two types of workflow:

- Sequential Workflow, and
- State Machine Workflow

We will use the state machine to have the ability to monitor the attacks from malicious sources effectively. The main advantages of the state machine are that it is not predictable when moving from one state to another and that it is driven by outside events.

A state machine is divided into a series of states and transfers from one to another through transitions which are generated by events. An example of an event is a user turning on the system: the system transfers from the "off" state to the "on" state. A state machine cannot be in two states at the same time; this makes it easy to query the state of the workflow. The basic idea of this project is to generate the state machine automatically according to the system state, and to build up a map of the system's states from the first interaction between the honeypot and the server until the end of the session.

The benefits of using an automated state machine with a high-interaction client honeypot are:

1. The state machine is generated automatically according to the system's states while it is working.
2. The state machine generated includes each action performed by any malware and saves the action along with its parameters.
3. For each client honeypot scan, the state machine is generated afresh.
4. Each state machine contains all the system states while interacting, which helps to review the system states.
5. The map of the state machine for each interaction helps to analyse the malware and identify its processes in order to develop a patch for it or learn its tricks.

There are different types of state machine, such as abstract state machines and the finite state machine we used. This was chosen because we want to monitor finite and limited aspects of the system, such File system, Registry, Processes and Ports. These are the four main states that we want to monitor and we will build our state machine on them. The state machine diagram is shown in Fig. 2. This finite state machine can be also illustrated with the 5-tuple as

$A = \{S, \Sigma, \delta, S_0, S_5\}$. Where:

- The Σ alphabet input includes the finite number of events $\Sigma = \{e1, e2, e3 \ldots, e18\}$.
- S is the none-empty finite states as $S = \{s_0, s_1, s_2, s_3, s_4, s_5\}$.
- S_0 is the initial state.
- S_1 is the File system state.
- S_2 is the Processes state.
- S_3 is the Registry state.
- S_4 is the Ports state.
- S_5 is the final state.
- δ is the transition function to states as

$$\delta: S \times \Sigma \rightarrow S.$$

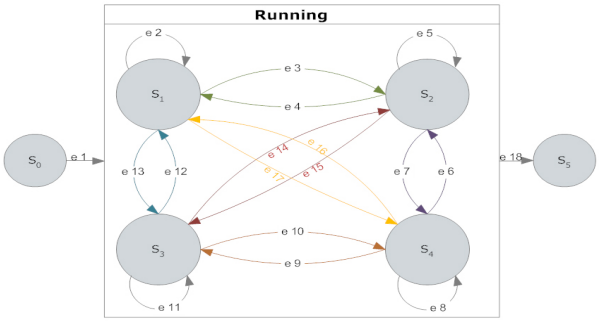

Fig. 2: The state machine diagram. (The boxes indicate states and the arrows represent transitions.)

The state machine includes four main states that will be monitored through the 18 possible transitions between the states. The finite state machine can be described as a state table which shows the possibility and direction of each transition to/from state, this table is shown in Table I.

TABLE I. THE STATE TRANSITIONS TABLE OF THE FINITE STATE MACHINE ON FIG 2

	e 1	***e 2***	***e 3***	***e 4***	***e 5***	***e 6***	***e 7***	***e 8***	***e 9***
S0	S1, S2, S3, S4, S5	θ	θ	θ	θ	θ	θ	θ	θ
S1	θ	S1	S2	θ	θ	θ	θ	θ	θ
S2	θ	θ	θ	S1	S2	S4	θ	θ	θ
S3	θ	θ	θ	θ	θ	θ	θ	θ	θ
S4	θ	θ	θ	θ	θ	θ	S2	S4	S3
S5	θ	θ	θ	θ	θ	θ	θ	θ	θ
	e 10	***e 11***	***e 12***	***e 13***	***e 14***	***e 15***	***e 16***	***e 17***	***e 18***
S0	θ	θ	θ	θ	θ	θ	θ	θ	S5
S1	θ	θ	θ	S3	θ	θ	θ	S4	S5
S2	θ	θ	θ	θ	θ	S3	θ	θ	S5
S3	S4	S3	S1	θ	S2	θ	θ	θ	S5
S4	θ	θ	θ	θ	θ	θ	S1	θ	S5
S5	θ	θ	θ	θ	θ	θ	θ	θ	θ

The transitions table explains the transitions between states, because the state machine is not like a Sequential workflow in which you can predict which transition will flow from/to which state. In the state machine you have to try to make a transition for every possible event that may occur while the computer is affected by a malware. For example, table 1 shows that the transition (e 14) will transfer the state machine from state S3 to S2, and that can also be seen in Fig. 2. Therefore, the system will remain in state S3 which is the state representing Registry, and if the malware tries to add a new process or kill an existing one then our automated state machine will transfer from the Registry state (s3) to the

Process state (S2) using transition (e 14), from this we can identify all malware actions and works. There are 18 transitions which cover all the possible actions that can be performed by the computer while infected by malware on the four main states: File system, Processes, Registry and port states.

As mentioned earlier, there are several issues to discuss concerning the finite state machine, as well as the client honeypot in order to gain a better understanding. Some of the issues, such as the system states to be monitored, the transitions from one state to the next, the information saved with each state, and the tools for monitoring changes in each system state, are worth mentioning here. These issues are discussed in the next section.

IV. A CLIENT HONEYPOT WITH A STATE MACHINE

The previous section described the state machine and its benefits for client honeypots. In order to incorporate an automated state machine into a client honeypot, the issues listed in section III are addressed below.

The **first question** is "What are the monitored system states?" To answer this question we have to understand the system states during infection by malware, and the other benign states. The system state can be changed by adding a registry value, running a new process, or inserting a new cookie. We also have to look at the tools that can be used to monitor each change. The system states that will be monitored while interacting with a target server are:

1. The File system will be monitored for any changes such as adding, deleting or modifying files or folders.
2. The Registry will be monitored for any added, deleted, or modified registry values.
3. Processes will be monitored for new processes run or processes started by the default process killed.
4. TCP and UDP ports will be monitored to detect if any ports are opened or closed; these ports could be used as a backdoor by the attacker.

The **second question** is "How does each state transfer to the next state?" Each of the four states listed above will have different transitions to another state. Each transition is responsible for transferring from the current to the next state or even returning to the same state. For example, malware can add a new file, which will transfer the system state to the file system state. If the malware then adds another file, the state will transfer to the same state, but with new information which adds the new file along with its action to build the state machine map. The transitions are described in Table I.

The **third question** is "What information will be saved for each state?" In each state there are several pieces of information that should be saved with the state to identify and show the malware activity. The information items saved with each state are:

- **Type**: This field saves the action which can be different for each state. Table II shows each state with its types.
- **Action**: This field saves the action parameters such as the target file created or the registry value inserted.
- **Source**: The source who cause the changes.
- **Destination**: The destination file, process or values has been change.
- **Time**: The time for each change to make up the state machine map ordered by time.

The **final question** is "What are the tools for monitoring system state changes?" The answer to this is fully described in Section V which shows the monitoring tools. Our main aim here is to build a framework that combines some free and open source tools with our tools to help us monitor the client honeypot effectively as well as allow us to customize them to our needs.

TABLE II. THE STATE TYPES

State	Type
File system	Add, modify or delete
Registry	Add, modify or delete
Processes	New or kill
ports	Open or close

V. MONITORING AND DETECTION TOOLS

Open source and free security tools have been developed over the last few years for the Windows environment and have lead to the improvement of Windows security tools and application. Consequently there are many security tools available for Windows, which will be the host operating system for our client honeypot. Table III shows each monitored activity along with the relevant tools. There are some tools such as the Capture-HPC [8] client honeypot, which can already monitor the file system, Registry and Processes, that could be a base for combine it with our or other tools to get the other activity that Capture-HPC does not support such s the ports. The framework that combines all the activities and the monitors tools to get the changes of each state and feed that to our automated state machine to transfer the machine state from one to another in the client honeypot. The idea of our client honeypot is to divided the client who interact with malicious website and the honeypot server which will store all the information about the client honeypot states and malicious activity and take control after finishing from scan a malicious website to reset to a clean state and then feed it with another malicious website to interact with.

TABLE III. MONITORING TOOLS

activity	File system	Registry	Process
tools	FileMon[3]	RegMon [6]	Process Monitor[5]
	FileSystemWatcher[4]		
	Capture-HPC[8]		
activity	**Ports**		
tools	Winsock Control [7]		

VI. DRIVE-BY DOWNLOAD ATTACK

A malicious webpage define as a web page that attempts to cause the automatic installation of software without the owner of the computer or authority knowing [11]. Then the malicious web page tries to attack the user, which poses a real danger because only a highly skilled user is able to detect the attack. Generally, the attacks are divided into two main forms:

1. **Server-side attack**: in a server-side attack, the server provides some service such as a web server which could have a vulnerability that leads to attacks on the service and the possible takeover of the whole server. For example: an attack on a MySQL database by SQL injection.

2. **Client-side attack**: in a client-side attack the attacker threatens the client application within the user environment. The client application has to interact with the malicious server in order to be attacked. For example: web browsers, media players and office applications.

Attackers have preferred client-side attacks in recent years because by the improvement of a wide range of security measures such as firewalls and IDS have made server-side attacks difficult. Therefore, the attackers (Blackhats) are attracted to client-side attacks because of the lack of security measures [12].

The goal of the attacker is to place malware into a user's environment and then collect important data such as credit card numbers, game usernames and passwords. The biggest risk is that banking login information could be stolen. The attackers are able to detect the vulnerabilities in a user's application from just a single visit to a malicious web page and then exploit this vulnerability to get control of a user's machine. This attack is called drive-by download. An example of a drive-by download attack will be shown in the next section.

A. An example of drive-by download attack

An example is shown below to understand how this attack works because is the main type that a client honeypot will face while interacting with servers.

1. The vulnerability. The vulnerability is in Internet Explorer, which is the default internet browser on the Windows operating system. The vulnerability is a DHTML Object Memory Corruption caused by race conditions on the memory management routines in DHTML objects in IE version 5.01, 5.5 and 6. The result of this vulnerability is that an attacker is able to execute malicious code remotely via his malicious web page as well as send a hazardous email with HTML capability. The vulnerability is called CVE-2005-0553 CVE ID and MS05-020 ID on a Microsoft security bulletin. According to Microsoft [12] the vulnerability is a remote code execution. "If a user is logged on with administrative user rights, an attacker who successfully exploited this vulnerability could take complete control of an affected system. An attacker could then install programs; view, change, or delete data; or create new accounts with full user rights. Users whose accounts are configured to have fewer user rights on the system could be less impacted than users who operate with administrative user rights".

2. The experiment. The vulnerability was tested in a virtualization environment to reduce the risk of a successful and damaging exploitation. The test operating system was Windows XP Service Pack 2 running Internet Explorer 6 with default settings. The experiment was monitored manually with the different security tools available. The author will do more experiments in the future using the honeypot technology. The tools that were used are:

- Active Ports 1.4: this software is a user-friendly and capable tool which shows the open ports on the machine along with their operating software in a graphical user interface (GUI).
- Process Monitor v2.03: the tool is a real-time monitor which can show each process with its work. The processes can be filtered to check the Internet Explorer's activities. The tool also shows all the entries for IE, additions, modifications and deletions of the registry or the file system for each process.
- Process Explorer v11.33: the tool is an advanced graphical interface used for real-time processes. This could help us to monitor the state of the processes while exploiting the vulnerability to see if there are any new processes or even run some new ones within IE.

3. The exploit. The vulnerability scenario is centered on the user making a single visit to the malicious web page with their vulnerable Internet Explorer (5.01, 5.5 and 6). The malicious web page in user's browser is shown in Fig. 3. As shown in the Fig. 3, the user will not notice anything different; rather he will just see a blank page. The malicious code will exploit the vulnerability in the background; a shell code is responsible for opening a backdoor for the attacker. The exploit is publicly available at [11]. The shell code can do much more than just open a backdoor; it can install malicious code on the victim's machine which will operate in the context of the user running Internet Explorer.

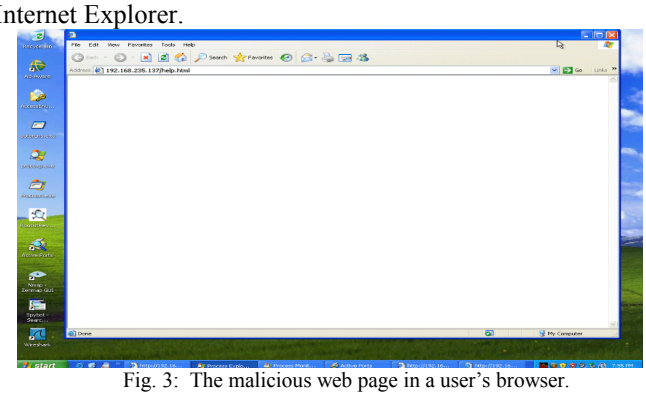

Fig. 3: The malicious web page in a user's browser.

4. The transfer. The exploit is hosted on a web page that the user can browse using a HTTP protocol application such as Internet Explorer. The transfer of this exploit is not complex, as is shown in Fig. 4. The transfer starts with SYN, ACK/SYN and ACK and then transferring the web page after receiving the GET request from the user for the help.html page; the reply to this request is web page code which contains the exploit. Once the user receives the content of the page and

views it, it will automatically exploit the vulnerability if the user has it.

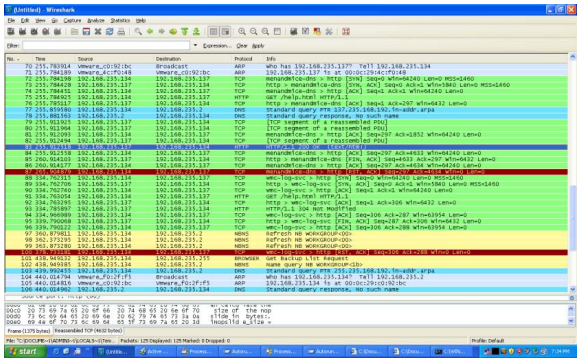

Fig. 4: The transfer of the exploit from Wireshark[10]

5. The result. By monitoring the state of the processes while opening the malicious web page we can identify a process that runs for just a few seconds before killing itself. That process has the name rundll32.exe, which is a part of the Windows operating system and helps to execute some command line provided. Fig. 5 shows the processes monitor in action.

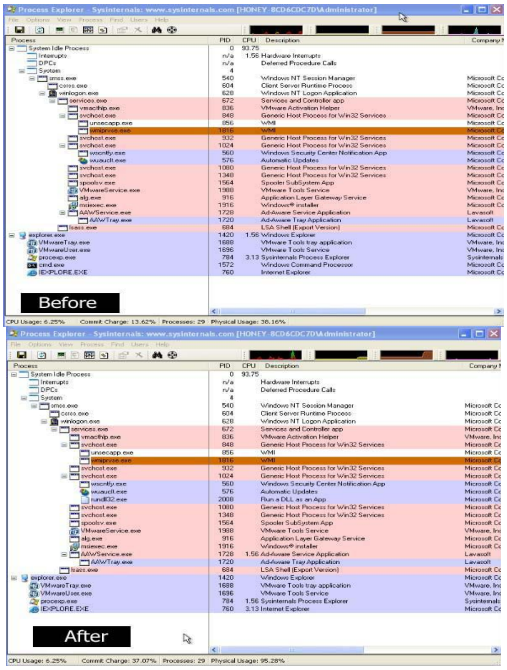

Fig. 5: The processes monitor in action.

After a successful exploitation from the attacker's point of view, the user will notice no difference in his computer or Internet Explorer's status. Port 28876 will be open for the attacker to connect remotely to the user's machine. The use of an active port shows that the port was opened by iexplore.exe which is the process name of Internet Explorer (IE). Fig. 6 shows the port in action after the exploitation of the vulnerability in the user's machine.

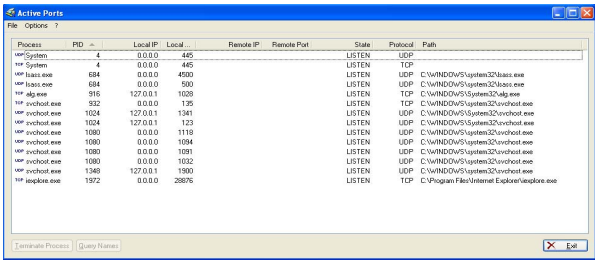

Fig. 6: Port activity after the exploitation of the vulnerability in the user's machine.

B. Automated state machine example

The previous section shows how the drive-by download attack can be detected by the security tools available in the Windows environment. The result of this attack scenario can also be shown in our state machine diagram to get a clear example of how the automated state machine could be applied in the high-interaction client honeypot. To summarize the previous attack for understanding its steps, so that we can apply it to the state machine, the steps are shown below in time order:

1. The exploit transfer via the source page to the user's vulnerable browser.
2. The attack exploit by starting a new process called rundll23.exe which was not run from the default processes.
3. The TCP port 28876 is opened from the new process running.

As shown above the steps are easy to understand and applied to the state machine. Fig. 7 shows the automated state machine of the CVE-2005-0553 exploit.

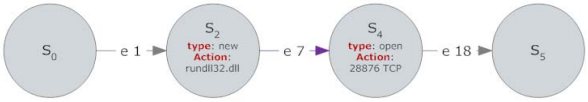

Fig. 7: The automated state machine of the CVE-2005-0553 exploit.

The Fig 7 above shows our automated state machine applied on the CVE-2005-0553 exploit; there are four states which summarize the exploit and the machine states. The states' steps are illustrated below:

1. **S_0**: The first state is the initial state as shows in the Fig. 7 above, the initial state started with (e 1) transition which is responsible to transfer the machine state from the first state (default state) to new state by adding new process. Therefore, the state gets transferred to state S_2.
2. **S_2**: The processes state according to the automated state machine in Fig. 2, the state described the new state of the machine and shows the new action which runs the 'rundll32.dll' process. The state then used the transition (e 7) to transfer the machine state to the next state which is S_4.
3. **S_4**: is the Ports state as described in Fig. 2. This state is transferred by the transition (e 7) and indicates that there is a new port open i.e. port 28876 TCP. Then, the state used the transition (e 18) to transfer to S_5 state.
4. **S_5** state is the final state which ends the exploit activity and effects to the system.

As mentioned above, by using the automated state machine we can save the whole changes map of an exploit and malicious software within the client honeypot. The automated state machine saves each exploit with its state machine map for further analysis and it can be used to understand some exploit effect such as most of the malicious activity that started by adding new process and then edit a particular file it will be finish by opening a particular port.

VII. EVALUATION

The evaluation section is focused on testing the state machine concept applied to a real client honeypot to see the benefits of our idea in a real situation and to determine how it can be improved in future. The evaluation section will be divided into three main sub-sections which are:

- **The client honeypot**: describes the client honeypot that will be used with the state machine and how it can be integrated.
- **The state machine**: describes the state machine's specification and its implementation.
- **The evaluation experiment and result**: shows the results of the state machine applied to the client honeypot.

Firstly, to test our idea of a state machine, we have to build or choose monitor tools that will interact with the system to get the system's state and changes over the time the system is being affected by malware. We have chosen Capture-HPC, [8] an open source client honeypot tool, which can monitor three main states: File system, Process and Registry. Capture-HPC will be the basis of our state machine; this means we will not monitor the port state in this experiment as it is not supported by Capture-HPC. We have received 26 log files from Capture-HPC from Dr. Peter Komisarczuk at Victoria University of Wellington; these log files will be passed through our state machine to convert them to the state machine log files used in our experiment.

Secondly, the state machine will build upon the Capture-HPC log files so we will understand the structure of the files, then our state machine can be built. Fig. 8 shows a snapshot of a Capture-HPC log file.

```
"file","Nov 11, 2007 2:50:50 PM","C:\Program Files\Internet Explorer\IEXPLORE.EXE","Write","C:\WINDOWS\TEMPmbroit.exe"
"process","Nov 11, 2007 2:50:50 PM","C:\Program Files\Internet Explorer\IEXPLORE.EXE","terminated","C:\WINDOWS\TEMPmbroit.exe"
"registry","Nov 11, 2007 2:50:50 PM","C:\WINDOWS\TEMPmbroit.exe","SetValueKey","HKLM\SYSTEM\ControlSet001\Control\Session Manager\PendingFileRenameOperations"
```

Fig. 8: shows a snapshot of Capture-HPC log file.

The snapshot shows three changes have been detected: write a file, terminate a process and insert a new registry value. For each change there are five elements which have to be detected:

1. **The state**: the location of the change; could be one of three: File, Process or Registry.
2. **The time**: the time of the change.
3. **The source**: the source of this change.
4. **The type**: the type of change, which is different from a state to another. File change types are: Write, Delete or Create. Process changes can be: Created or Terminated. Finally, registry changes can be: Set value or Delete value.
5. **The destination**: shows the destination of the effect caused by the source file which can be a file, process or registry value.

After understanding the log file, the state machine can be built upon this knowledge. The state machine will be built with Windows Workflow Foundation [14] and will monitor three states for this experiment: File system, Process and Registry. The architecture of our state machine is shown in Fig. 9 where:

- **S0**: the start state.
- **S1**: the File system state.
- **S2**: the Process state.
- **S3**: the Registry state.
- **S4**: the completed state.

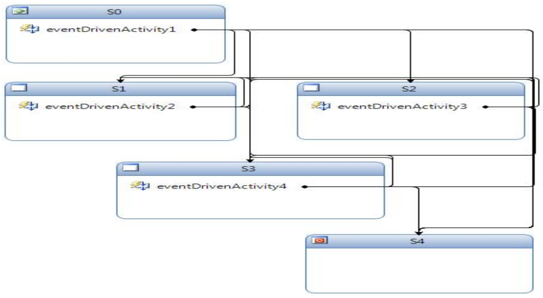

Fig. 9: the state machine states in Windows Workflow Foundation.

After building the state machine and choosing Capture-HPC to be our base monitor tool, we now need to test the tool and gather results. As mentioned before, we have received 26 different log files produced by Capture-HPC; each file shows a scan result for a different malicious web page. Our state machine tool can be configured from an external configuration file making it easy to configure it to convert a capture-HPC log file to a state machine version; the configuration file also can set the folder name of the conversion process. Fig. 10 and 11 show the state machine tool in action and the result files.

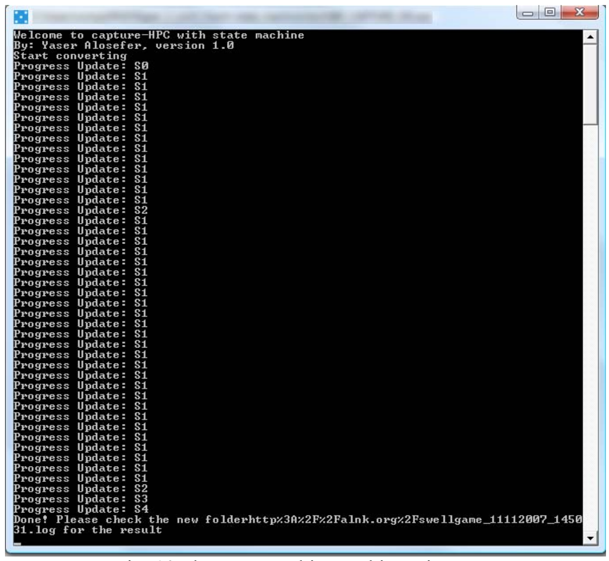

Fig. 10: the state machine tool in action.

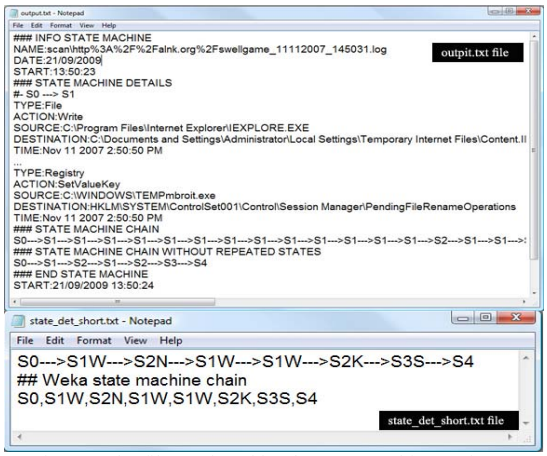

Fig. 11: The file produced by the state machine tool.

The results can be seen in Fig. 11 above. The file output.txt is the main state machine file which contains all the states changes; at the end of the file are two state machine chains, the first shows the states of the log file with repeated values, while the second shows the states without repeats which makes it easier to see and figure out the malware changes on the system. The second file in Fig. 11 is the state_det_short.txt file which contains the state machine chain without repeated values; most malware will try to make changes more than once to ensure it achieves its aim. The state machine chain also shows one character which represents the change type with the state; for example, the state machine chain from Fig. 11 is shown below:

S0--->S1W--->S2N--->S1W--->S1W--->S2K--->S3S--->S4

The state machine above shows the state and type of change in each stage, the states listed above are the same states described in Fig. 9, while the characters after each state name are described below:

State	Character / Describe	
File system (S1)	W	Write
	D	Delete
	C	Create
Process (S2)	N	New
	K	Kill
Registry (S3)	S	Set
	D	Delete

The evaluation experiment was to convert all the client honeypot log files to state machine one, and then to use the state machine log files to understand something from the log files as an analysis engine. We have converted all 26 log files to the state machine version as the example shows in Fig. 11. We then used the Weka data mining software [15] to get an over view of our 26 state machine log files in a graphical format and to understand some facts which help the analyser. The Weka software supports files such as ARFF (Attribute-Relation File Format) files, our state machine tool produces a Weka state machine chain that can be fed to a pre-defined ARFF template made by us and then the file can be imported to Weka to get interesting results. Fig. 11 shows the output results from Weka after feeding them into our state machine log files. Fig. 12 shows the graphical output of all 26 state machine log files with some modifications to explain the results to the reader. The figure also shows the first six stages of each malicious web page state change. Some of the results from Weka can be summarized as below:

- The first stage shows that all 26 malicious web pages are to write a new file to the system.
- The second stage shows some differences: 23 malicious web pages write another file to the system while three launched new processes.
- The third stage illustrate two malicious web pages writing another file to the system, while most launched new process and only one malicious web page inserted a new value into the registry.

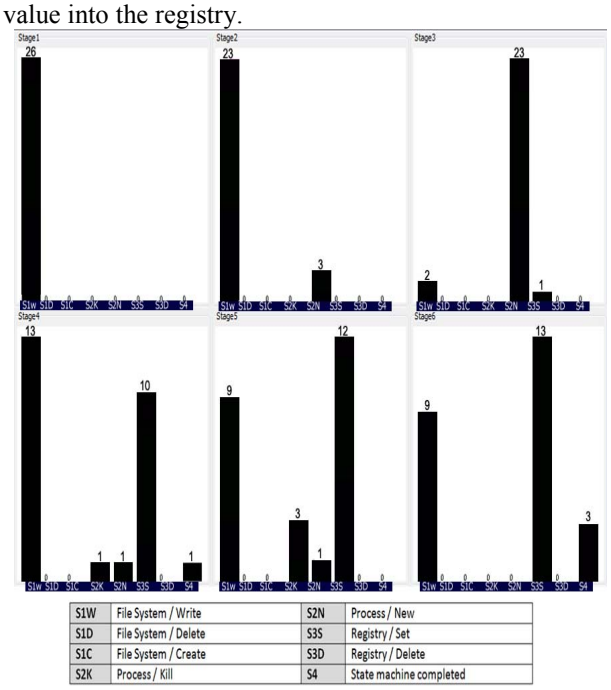

Fig. 12: shows the Weka graphical output of the 26 malicious web pages.

The experiment results are shown in Fig. 12 using a simple output from Weka. The same Weka file can also be used by Weka to get advanced results using data mining algorithms, such as clustering algorithms.

VIII. FUTURE WORK

The current state of this work is apply an automated state machine to a client honeypot to get a map of the changes for further analysis by different groups of people such as researchers, security analysts, or other organizations. This is the first step to build up a complete architecture that improves how the client honeypot works and deploys. The current status of state machines and client honeypots needs some improvements to be more useful and useable such as:

- **The state**: the current states that supported by the state machine are four main states as shown in Fig 2. More states can be added to get further information and whole system

monitoring for any other activity by malicious scripts could be done. An example of state that could be added to a future state machine are Network connection, the check of a mutex or Windows services.

- **The Process Follower:** If a malicious website is trying to infect the user with different malware, it could happen that the user will be affected by different malware at the same time. How could a state monitor know if the file system changes were caused by malware A or B? There has to be a way to distinguish each malware and then follow its changes so that there is no confusion about the activities of malware and we get to know the correct results for each malware along with their changes and activities.
- **Improve the Client honeypot**: The client honeypot tools available such as Capture-HPC has a good way to detect the malicious website. However, many more malicious websites are trying to avoid being detected by honeypot. For example, a malicious website could require clicking on the website to lunch the malicious website because all the current client honeypot are just visiting the website and then scanning and monitoring the system for any changes. To improve that, the client honeypot can customize the Internet Explorer which is the default browser that interacts with the malicious website and adds new toolbar. This toolbar can help us to click on the page when visiting it or even auto filling and submitting any forms on the page to get a better chance for catching any malicious technique that remains hidden from the existing honeypot.
- **The stat machine chain**: the state machine chain used here is simple and it is affective while analysis a single file as well as group. In future, we will improve the state machine chain that will contain the source and target paths for each event which will be very powerful and meaningful when compare and track the state machine events by in the same file or with others. The new state machine is shown below:

```
S0---> S1W[C:\Program Files\Internet
Explorer\IEXPLORE.EXE]=[C:\WINDOWS\TEMPmbroit.exe]--->
S2N[C:\Program Files\Internet
Explorer\IEXPLORE.EXE]=[C:\WINDOWS\TEMPmbroit.exe]--->
S2K[C:\Program Files\Internet
Explorer\IEXPLORE.EXE]=[C:\WINDOWS\TEMPmbroit.exe]--->
S3S[C:\WINDOWS\TEMPmbroit.exe]=
[HKLM\SYSTEM\ControlSet001\Control\Session
Manager\PendingFileRenameOperations]--->S4
```

An explanation of second event to get a better understanding about the new state machine chain: (S1W [C:\ProgramFiles\InternetExplorer\IEXPLORE.EXE]=[C:\WINDOWS\TEMPmbroit.exe]--->) as follow:

1. **S1**: change in file system state.
2. **W**: the type of change is write.
3. **C:\Program Files\Internet Explorer\IEXPLORE.EXE**: the source of change.
4. **C:\WINDOWS\TEMPmbroit.exe**: The result from this change, which is writing the TEMPmbroit.exe to the file system by Internet Explorer.

IX. CONCLUSION

The client honeypot is an excellent tool for use in the security industry that helps to trap and understand malware in depth.

The state machine tool that has been developed to evaluate the state machine concept in the evaluation section used Capture-HPC as the client honeypot engine. The next stage is to improve the client honeypot which will add support for port states so it can monitor more state machines which the malware can use to open or close ports, for example to send notifications or system information. Another approach would be to improve and use analysis algorithms to increase the benefits of state machine use.

The automated state machine, when it is applied to a client honeypot as described in this paper, gives honeypot technology a number of benefits when analysing malware and tracing the details of malware steps. There is a need for an open source framework to combine the benefits of all the tools mentioned in this paper with the use of the automated state machine.

X. REFERENCES

[1] Spitzner, L. 2002. Honeypots: Tracking Hackers. 1 edition. Addison-Wesley Professional. ISBN-10: 0321108957.

[2] Allen, K. Scott. 2006. 'Programming Windows Workflow Foundation: Practical WF Techniques and Examples using XAML and C#'. Published by Packt Publishing Ltd. ISBN 1904811213, 9781904811213.

[3] Russinovich, M, Cogswell, B,. 2006. FileMon for Windows v7.04. Available at: http://technet.microsoft.com/en-us/sysinternals/bb896642.aspx [Accessed 11 Feb 2009]

[4] .NET Framework Class Library. FileSystemWatcher Class, Available at: http://msdn.microsoft.com/en-us/library/system.io.filesystemwatcher(VS.80).aspx [Accessed 11 Feb 2009]

[5] Russinovich, M, Cogswell, B,. 2009. Process Monitor v2.04. Available at: http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx [Accessed 11 Feb 2009]

[6] Russinovich, M, Cogswell, B,. 2006. RegMon for Windows v7.04. Available at: http://technet.microsoft.com/en-gb/sysinternals/bb896652.aspx [Accessed 11 Feb 2009]

[7] Microsoft Help and Support. 2004. How To Monitor TCP/IP Ports in Use. Available at: http://support.microsoft.com/kb/194938 [Accessed 11 Feb 2009]

[8] Capture-HPC. 2008. Available at: https://projects.honeynet.org/capture-hpc [Accessed 11 Feb 2009]

[9] Wireshark. Available at: http://www.wireshark.org/ [Accessed 11 Feb 2009]

[10] SkyLined. 2005. Internet Explorer DHTML Arbitrary Code Execution (MS05-020). Available at: http://www.securiteam.com/exploits/5CP0C0UFGG.html [Accessed 11 Feb 2009]

[11] Seifert, C, Steenson, R, Holz, T, Yuan, B, & Davis, M,. 2007. Know Your Enemy: Malicious Web Servers. [Online]. Available at: http://honeynet.org/book/export/html/153 [Accessed 11 Feb 2009]

[12] TechCenter Security. 2005. Microsoft Security Bulletin MS05-020. Available at: http://www.microsoft.com/technet/security/bulletin/MS05-020.mspx [Accessed 11 Feb 2009]

[13] Seifert, C,. 2007. Know Your Enemy: Behind the Scenes of Malicious Web Servers. [Online]. Available at: http://honeynet.org/book/export/html/181 [Accessed 1st Mar 2009].

[14] Windows Workflow Foundation. [Online] Available at: http://msdn.microsoft.com/en-us/netframework/aa663328.aspx [Accessed 5 Aug 2009]

[15] Weka 3. Weka 3: Data Mining Software in Java [Online] Available at: http://www.cs.waikato.ac.nz/ml/weka/ [Accessed 12 Sept 2009]