This report is a detailed overview of the work we did while building and deploying a honeypot. A number of tools and methods we tried weren't successful enough to be used in the final deployment. The final report of this project sent earlier was strictly focused on the final deployment and results we obtained.

We were focused on setting up a Honeypot network based upon well-adapted methods and were successful in partially establishing them.

Books on Honeypots

We started the project by going through the most recent literature available on honeypots. The resources were the papers available on the website honeynet.org and the book "Virtual Honeypots". There were other books on honeypots but this one was the most recent. The book helped us gain a general understanding of honeypots - types, a brief history, legacy methods, current methodologies etc.

It involved setting various kinds of honeypots, including a tool honeyd, specialized in honeypot deployment.

Honeyd

Honeyd is a low interaction honeypot to instrument some unused Internet addresses with virtual honeypots and corresponding network services on the honeypots. For e.g. we could create a virtual honeypot on an IP address with a network stack that looks like Windows on which TCP ports seem to be running services. In this case, it would allow us to receive the first TCP payloads for worms of probes. To provide a realistic honeypot to the attacker, honeyd even deceives fingerprinting tools.

After deploying honeyd, while creating virtual honeypots and and some services to run on them, we found that it simulates very old versions of Operating Systems and services. This made us to go for setting up a high interaction honeypot where recent OS and services are running.

SSH-Patch

In our early meetings with you, we discussed about the possibility of logging the shell activities of an attacker. While looking for solutions, we came across a tool 'Script', which logs everything that goes on in a shell. We configured the bash shell such that every time a new bash session is opened, script is activated and starts saving the shell's state in a hidden file.

Now, the next target was to find a way to give an attacker the shell access. We started looking for applications which had a shell execution vulnerability. It didn't turn out to be a good solution in our case. To begin with, we're looking for server based vulnerabilities, since the honeypot was going to play the role of a server. And all the sources that we managed to find with such a vulnerability we're too old to be used by an attacker now. As a result, we decided to go for an

easier way, ssh access.

A lot of attackers use malicious scripts which keep on brute-forcing the ssh servers of live systems. We planned to create common users on our system with simple passwords and may be even a guest account with none. This would have resulted in an easier and unsuspected access to the attacker.

Since, its a honeypot and we're interested in gathering as much attack data as possible, we even patched the openssh server to log all the passwords attempted.

This would have been a decent solution for deployment too, but the trouble was the data collection. It wasn't wise for a honeypot to keep all the logged, vital data on the same system as the attacker. It wasn't a very stealthy method and could have been easily caught by a careful attacker and before we could have anticipated the situation, the data could be gone.

We opted for sending the data to a remote log server but the communication would have been easily viewable to the attacker. At this point we started browsing through the different honeypot technologies used over time and found out about two major tools specialized in collecting intrusion data through honeypots, Sebek and Qebek.

Sebek and Qebek

Sebek is a linux kernel module which slightly modifies the functioning of the kernel as per its requirements. Its a data collection tool for high interaction honeypots. This project is no more under development. Qebek, on the other hand, is a next generation data capture tool which aims to stealth the whole process of monitoring and data collection by residing outside the OS(it currently supports only Windows), in the Virtualization layer.

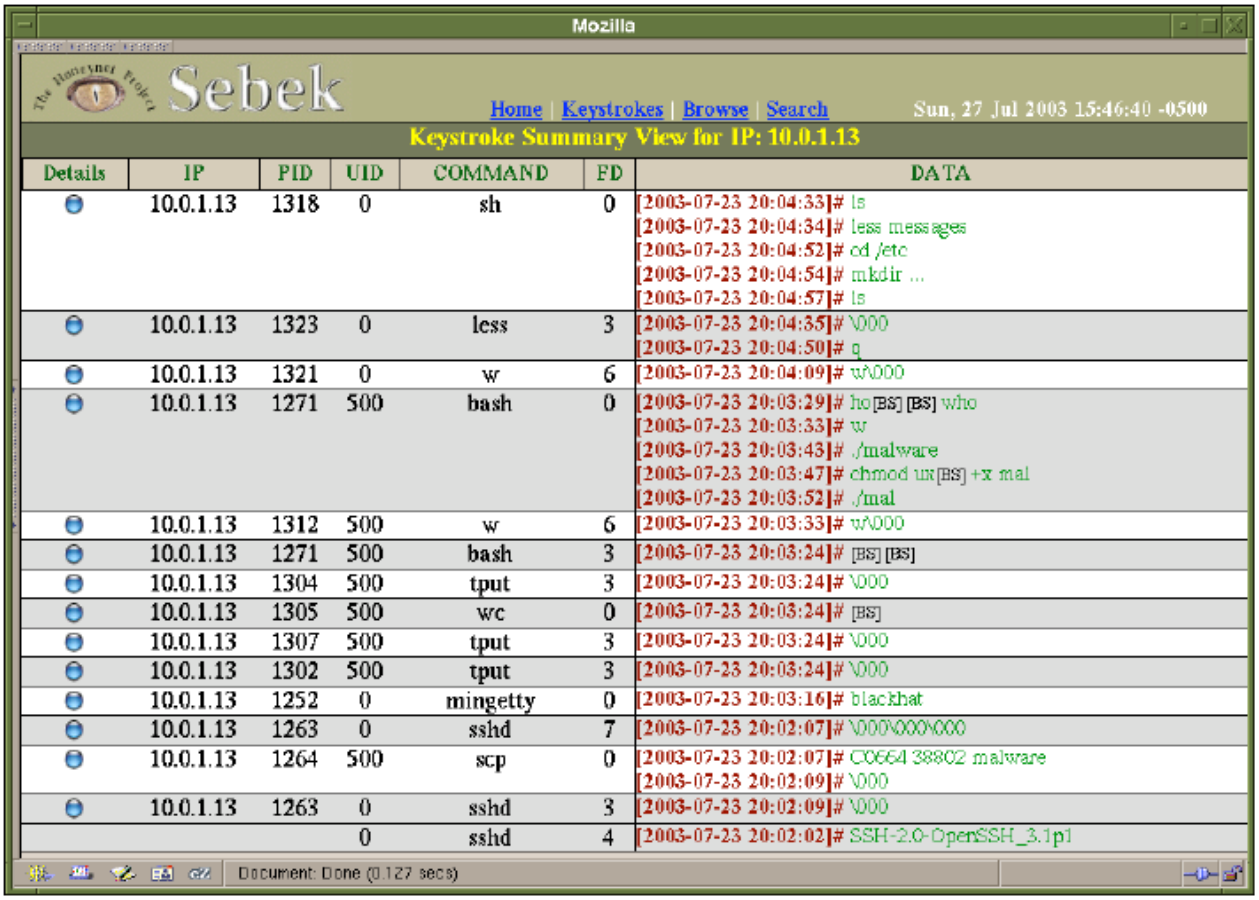
Sebek uses a client-server architecture. The client, sebek, is installed on the honeypot and the server, sebek daemon, is installed on a separate system for data collection. Sebek daemon comes with its own sets of tools which let you either dump the data on the console or save it in a database.

Our initial work was focused on getting the sebek client and daemon installed and setup for demonstration. The catch was the narrow range of Linux kernels compatible with Sebek module. As a result, we had to settle for a few options for the Operating systems we're going to use for deployment. It was working on Ubuntu 7.10 and 8.04. With 7.10 left unsupported and 8.04 being the long term release, we decided to use latter for the deployment.

Once we had sebek setup, we showed a working demonstration to you in February. We were still using the console for the data output. Clearly, this wasn't a good way of sharing results. As mentioned earlier, there is an option to save the data in a database, but it still needed an interactive web interface to extract all the results.

Sebek web interface

In Sebek's paper, there was a brief overview of the web interface that Sebek ships with. It seemed to be a perfect solution in the beginning apart from the fact that the repository available on the web doesn't have any trace of a web interface. Our understanding is that the current repository is a result of importing the source from the old Sebek repo, around 3 years ago. The most recent versions were imported and by then the web interface solution was shifted to wall-eye, a web viewer available in the Honeywall.



Honeywall

In order to the get web interface walleye, our next target was the successful setup of Honeywall. Its a customized CentOS build specifically to act as a monitoring hub for honeypots deployment. Its configured to act as a transparent bridge between the attacker and the honeypot. This way we can monitor all sorts of communication going in and out of the honeypot.

The earlier effort in Honeywall setup was troublesome. We couldn't even get the interfaces up for basic networking activities. Hence, setting it up as a transparent bridge, installing the

sebek daemon and walleye, configuring it to act as a gateway between outside world and our honeypot seemed time consuming.

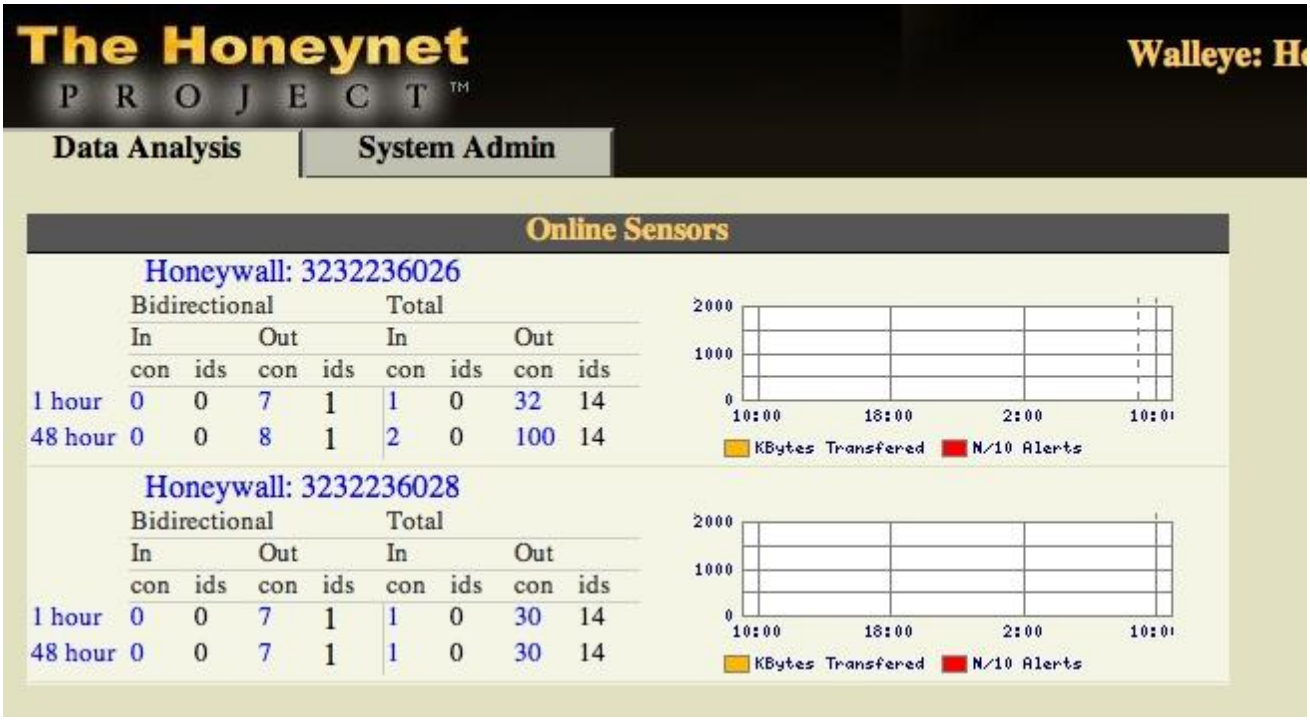
In order to quicken things, we decided to extract just the feature we wanted, walleye. Since, this was a customized CentOS, walleye was available in the form of a rpm package. But the installation wasn't straight forward. Walleye had numerous dependencies and many of them were libraries built specifically for controlling the internals of honeywall. These packages were not available in a regular CentOS built and installing them manually through the packages extracted from honeywall's ISO warned about serious version conflicts with core libraries.

This work around with Walleye was not successful. On a second attempt with Honeywall, while going through the configuration, we found out that it stays in a closed mode with firewall shutting down all the communication, hence no networking. We're successfully able to set it up in a week and got both sebek and walleye up and running.

Testing it a few times helped us realize that it doesn't provide the same features as Sebek and most importantly we can't monitor the shell activity logged by sebek using it.

In spite of the failure, the positive side were the features it provided. Walleye provided a powerful interface to monitor all the network activities that were going through the honeywall, the source and destination of any packet. The interface was well designed to view data through various perspectives

- all communication between honeypot a specific IP
- all communications on a specific port
- all communications over a period of time through the honeywall, and
- best of all, it even stores the contents of the packet over a period of time which could have helped us in capturing malware sent to the honeypot.

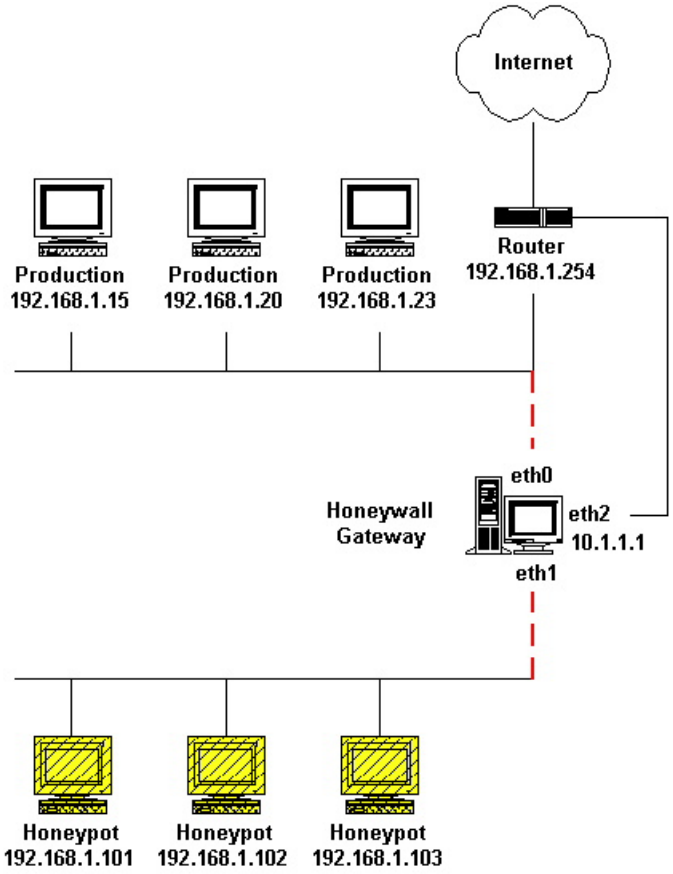


Even though Honeywall didn't provide the expected results, it stores complete packets which could be used for malware capture and analysis. Hence, we starting working on Qebek.

Qebek

Qebek is Qemu based Sebek, for Windows. Unlike Sebek, this data monitoring tool resides in the Qemu virtualization layer and logs all the activities on Windows from outside. The communication architecture is similar to Sebek, client-server. A regular Qebek installation will send the results to the configured recipient, running sebek daemon, which will be honeywall in our case.

By now, we're planning to deploy two honeypots, one running Linux and the other Windows. Our initial literature study suggested that such multiple honeypots deployment, aka honeynets, are done in a well-established method of using a single honeywall on the gateway. Since, we already had one configured we decided to go for this single honeywall method.



During the testing, when we connected Qebek to honeywall and viewed the data being captured by Sebek, we found that it was saving garbage values in the tables. Our understanding is that it was because of the protocol differences implemented in sebek daemon and Qebek.

Even though this was independent of walleye monitoring, we needed the data sent by Qebek. It had all kinds of vital information regarding the internals of Windows which would have been helpful in analysing the attack path/pattern taken by a malware.

Our initial plan of using walleye as the data viewer was unsuccessful and so was the setup of Qebek. We had already lost a good amount of time and hence we decided to put our wish of Windows monitoring on hold and concentrate on getting the primary aim of setting up a Linux based high interaction honeypot successful.

The above efforts helped us in understanding the proper methods of Honeynet deployment.