INTERCEPT: High-interaction Server-type Honeypot based on Live Migration

Daisuke Miyamoto
Information Technology Center
The University of Tokyo
2-11-16 Yayoi, Bunkyo-ku,
Tokyo, JAPAN
daisu-mi@nc.u-tokyo.ac.jp

Satoru Teramura
School of Engineering
The University of Tokyo
7-3-1 Hongou, Bunkyo-ku,
Tokyo, JAPAN
s.teramura@cnl.t.u-tokyo.ac.jp

Masaya Nakayama
Information Technology Center
The University of Tokyo
2-11-16 Yayoi, Bunkyo-ku,
Tokyo, JAPAN
nakayama@nc.u-tokyo.ac.jp

ABSTRACT

This paper aims at developing a honeypot system for web applications. The key idea is employing migration techniques to create a virtual machine as a honey web server, and making the honeypot to equip the same memory and block content of the real systems. Recently, web applications have been the target of numerous cyber attacks. In order to catch up new vulnerabilities in the applications, using a honeypot system is a feasible solution. However, it might be difficult to develop the lure-able, protect-able, and deception-able honeypot for web applications. This paper analyzes the background issues of the problem and finds the missing piece toward the suitable honeypot. It also designs and implements INTERCEPT, the core component of the honeypot system for web applications, which can avoid the data corruption as well as finishing the migration in short time period. Finally, we discuss how to complete the missing piece.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques

General Terms

Security, Emulation

Keywords

Honeypot, Web application, Live Migration

1. INTRODUCTION

Web applications have become one of the popular targets for cyber attacks. This is due to several reasons; for one, the web applications manage a wide array of information including financial data, medical records, social security, therefore attacks aim at stealing the information [20]. Another reason provided by McAfee [2] is the ease of exploitation of web vulnerabilities, combined with the proliferation of low-grade software applications written by inexperienced developers. According to the report from OWASP [18], web vulnerabilities allow a remote attacker to execute injection attacks, e.g., SQL injection [13] and/or Cross Site Scripting [7] attacks that input malicious codes to pose web hijacking, information leakage, malware infection, and so on.

A web application firewall (WAF) is one of the solutions against attacks which filters suspicious code injection. Generally, it inspects the application layer so it usually comes as an appliance type or as a server module [10]. There are several algorithms for distinguishing suspicious codes from HTTP request. The one is rule-based filtering, which checks the HTTP request with the database of known attack patterns. Besides the protection via blacklisting, WAF usually supports whitelisting; With active whitelisting, the rule set of the WAF describes the exact behavior of the application [8]. Different from the rule-based methods, heuristics-based methods calculate the likelihood of being a malicious code and compare the likelihood with the defined discrimination threshold.

Unfortunately, it still remains challenges in order to block code injection attacks. The rapid development of web applications arises numerous program bugs, the cause of web vulnerabilities; therefore, building a perfect blacklist is tedious work. In the case of the whitelisting, the issue is that it requires WAF operators to maintain rules of the legitimate input and output for the applications. The core issue in heuristics-based methods is detection accuracy. To achieve higher detection accuracy, monitoring injection attacks and analysis of them are necessary.

Herein, we present a server-type, honeypot systems for observing web attacks. Theoretically, honeypots are decoy systems to gather information regarding an attacker, and have deception and luring capabilities. A honeypot offers some services that appear perfectly normal to the attackers, and looks alike as if the system has *honey*, e.g., valuable data. Our motivation is to collect the information related to cyber attacks toward web applications using the honeypot.

Our developed *INTERCEPT* is the core component of the honeypot for web applications. The key idea is employing live migration technique; INTERCEPT creates the perfect copy of the legitimate virtual machine in few seconds and isolates the attacks to the honeypot. By orchestrating IN-

EMUTools 2014 March 17-19, Lisbon, Portugal.

Simutools 2014, March 17-19, Lisbon, Portugal

DOI 10.4108/icst.simutools.2014.254849

Table 1: Interaction Level

Interaction	low	high
Actual OS / applications	no	yes
Risk	low	high
Operational cost	low	high
Performance	low	high

TERCEPT and existing components, this paper describes the suitable honeypot which can protect valuable data from leakage as well as observing attacks.

The rest of paper is organized as follows. Section 2 briefly explain related work, and section 3 designs the architecture of the honeypot for web applications. Section 4 implements INTERCEPT, our proposed honeypot, and section 5 shows the preliminary evaluation of INTERCEPT. Section 6 discusses the missing piece toward the suitable honeypot, and finally summarizes our contribution in section 7.

2. RELATED WORK

Honeypot systems can be categorized into four types, namely (i) high-interaction client-type honeypot, (ii), low-interaction client-type honeypot, (iii) high-interaction server-type honeypot, and (iv) low-interaction server-type honeypot. The type, server or client, means that the honeypot systems work as server or client computer. The key characteristics of the interaction was explained in many articles [11, 12, 19], and the summary is shown in Table 1.

The client-type honeypot systems are used to discover new vulnerabilities in the client side systems, such as client OSes, web browsers and their plugins. The honeypot systems usually crawl suspicious web content, allow malicious content to exploit the system, and observe the attack methodologies. For examples of the type (i), HoneyClient [14], CaptureHPC [26], and Marionette [1] are used to analyze malicious URLs for blacklisting. Assuming if a system, which was composed of the latest version of Windows, Internet Explorer and plugins, was compromised while browsing. It can be assumed that there was new vulnerability in the system and the honeypot succeeded to find it. Instead of using the actual systems, the type (ii) honeypot systems such as Honey-C [22] and Monkey-Spider [27] mimic the client OSes, IP stacks and applications. While the attacks target particular applications, it can reduce the risk for compromising. However, the observable information during the attacks tends to be limited in comparison to the high-interaction honeypot, as shown in Table 1.

The server-type honeypot systems aim at obtaining new vulnerabilities by monitoring the trials of attack, penetration, and intrusion. In order to collect these events, the honeypot systems need to induce the malicious people to be targeted of the attacks. Since the type (iii) honeypot systems are faced to the risk for compromising, the abilities including containment are necessary. Referring to [24], we will explain the requirements of the honeypot systems. In the case of type (iv), the honeypot systems run tools aiming at emulated vulnerable systems. For example, Nepenthes [3] and Deception Toolkit [6] mimic the vulnerable applications. Unfortunately, there are several tools [15, 29] for identifying the systems remotely, therefore, the attacker are easily aware that their targeted system is honeypot.

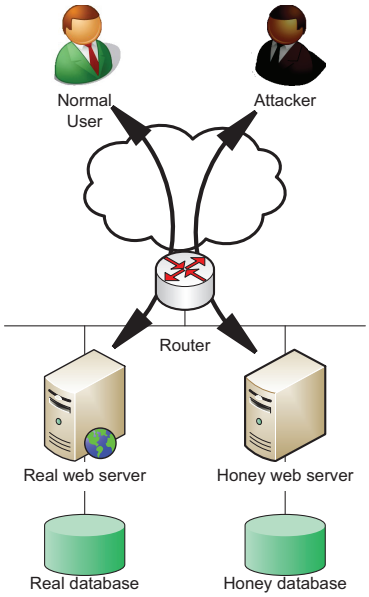


Figure 1: Concept of High-interaction Server-type Honeypot for web applications

3. DESIGN OF HONEYPOT FOR WEB APPLICATION

Our grand goal is to develop a novel honeypot system for web applications. In order to discover new vulnerability of web applications, honeypot systems facilitate to collect new attack methodology.

Through the survey described in section 2, we define the following abilities that honeypot systems for web application should equip.

- Luring ability

  Due to the nature of honeypot, the system must be attractive for attackers.

- Protection ability

  The system must equip protection capability against the attacks. Containment must be work for preventing the abuse of the systems.

- Deception ability

  The system must have deception capability, in other words, attackers cannot distinguish that their targeted system is honeypot.

In the context of the honeypot systems for web applications, we decide to employ high-interaction server-type honeypot systems. Since web attacks often require to login to web applications before their execution, the low-interaction honeypots are required to emulate many web applications. It might be tedious, because this task is equal to make the clone of the targeted application.

This paper designs our honeypot system as shown in Figure 1. In order to meet the luring requirements, it employs the actual web applications. A router normally directs to all web requests to the real web server. However, when suspicious web requests are detected, the router directs to the

requests to the honey web server. This can meet the protection requirements hence the attacks must not be harm to the real web server.

The most difficult point is that a honey web server must have the same state to a real server. In other words, the honey server is required to have the same memory of the real server regarding to the login state, as well as files of web content including applications. Assuming if the attacker needs to login to web applications before his attacking. Since his login process is not different to the normal one, this procedure might be done in the real server. After completing login, he sends suspicious requests to the system, and the requests are directed to the honey web server by the router. To deal with such request for the honey web server, the server must know his login state as same as the real server. Otherwise, the honey server may return some error messages to the attacker; the attacker can have a chance to be aware of that they are quarantined to the honey systems. It cannot meet the deception requirement.

In order to meet the deception requirement, we decided to experiment with migration techniques. They enable a virtual machine to be physically moved from one physical machine to another in a transparent way [5]. It also enables to create the complete memory and storage copy of the real web servers.

While using the migration technique to the honeypot, there still remains such problems that *determination of suspicious requests*, *direction of suspicious requests*, *prevention of data leakage*, *avoidance of data corruption*, and *migration in short time period*.

For identifying suspicious requests, the likelihood of being an attack, which can be calculated by the heuristics-based WAF, is useful. This is just a case, but if there is slight possibility of the attack, the router in Figure 1 can direct to the honey web server. Even the normal transaction is directed to the honey server, it must not be lost because the honey server can observe the transaction. Due to the honey server has the same state of the real server, it must not penalize normal users' convenience.

Since the migrated virtual machine has the same mac and IP address of the real servers, it must be considered for the direction of the suspicious request. Aside from the traditional routers, the modern packet forwarding systems such as OpenFlow and/or LISP routers can forward requests regardless of addresses. Besides, LISP is available for directing attack traffic to decoy server [21]. In corporation with the heuristics-based WAF, it can forward suspicious requests to the honey web server.

In order to prevent data leakage, it can be considered to separate databases for real and honey web servers. The real database has entire tables and records, and the honey database has limited tables and records that are related to the user who logged in to the web applications. While the honey database should not contain the records to other users, the risk of data leakage might be thwarted.

The missing pieces are *avoidance of data corruption* and *migration in short time period*. This paper focuses on development of the system in corresponding to these two problems.

4. DEVELOPMENT OF INTERCEPT

In our study, we setup Ubuntu 13.04, Kernel-based Virtual Machine modules to two physical machines; their specifications are shown in Table 2. We have modified QEMU [4]

Table 2: Specification of Physical Machines

	Sender PC	Receiver PC
CPU	Intel(R) i7-3610QM 2.30 GHz	Intel(R) i5-2450M 2.50 GHz
Memory	8 GB	4 GB
Disk	1 TB ATA	500 GB ATA
NIC	RealTek RTL-8169 (Gigabit Ethernet)	RealTek RTL-8169 (Gigabit Ethernet)

source codes. Due to the nature of migration techniques, a migration-source VM will power off to prevent a possible data corruption. Whenever our implement continues to run a source VM after migration, both suppression of the state change in the VM and prevention of the data corruption are necessary.

In order to keep source VM running after migration, we modified the state of source VM will be changed when `vm_stop_force_state()` and `runstate_set()` are called via `migration.c`. It works fine, so the rest of problem is the data corruption.

At first, we used Full Live Block Migration (FLBM) for preventing data corruption. FLBM supports the entire disk copying, therefore, the source VM and destination VM have respectively their own virtual disk images. It enables that the source and destination VM have the same content in entire memory and block devices as well as avoiding data corruption. However, FLBM requires a lot of time due to the copying full disk images during migration procedure. It might not be good solution for honeypot, because an attacker would feel something is not normal.

Different from FLBM, it was feasible to use of Incremental Live Block Migration (ILBM). In the "incremental" mode, only the blocks that were modified are migrated. QEMU's disk image utility supports to create a snapshot image file. Instead of FLBM, ILBM can dramatically reduce the time during live migration, but it still needs couple of time; it can be assumed that these times are required for verifying disk consistency and for memory migration.

In order to reduce the time for verifying disk consistency, we tested LiveBackup [25] and DriveBackup [28]; the former enables the destination VM periodically polls the source VMs to the data which might be backup, and the latter enables to push the data from the source VM to the destination VM. However, these tools are designed for backup, due to that they only work when the source VM is surely powered off. If the source VM kept running, we confirmed that the data corruption or critical system failure occurred.

Instead of using backup tools, we observed that combination of NFS and advanced multi layered unification filesystem(AUFS) [17] worked fine. AUFS is a implementation of Union File Systems, which provides the function, called branch, to unite several directories into a single virtual filesystem. Based on the solution, we conducted our experiment as follows.

1. Setup snapshot image files
   In order to reduce the further copy-on-write process, we decided to use snapshot. Besides to this, the source physical machine (PM) and the destination PM have the same base image file for the snapshot image file, and use the base image file to the same directory path in the physical machine. For example, if the destina-

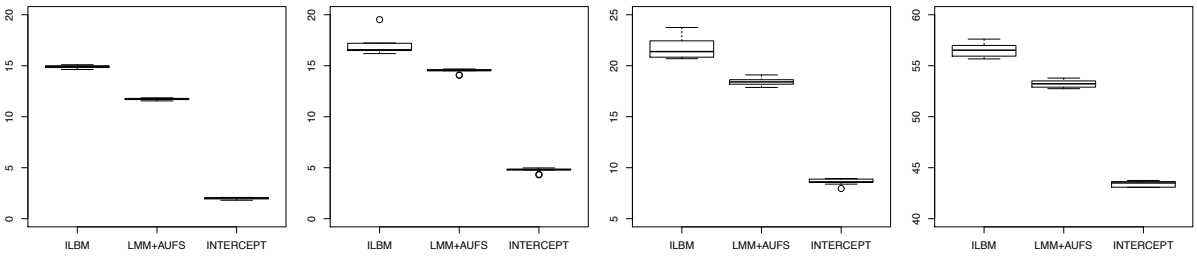
```
# ./qemu-system-x86_64
 -drive file=/nfs/debian-kvm001.qcow2,if=virtio
 -boot d -enable-kvm -monitor stdio -vnc :0
(qemu) migrate -d kemari:tcp:192.168.1.2:4444
```

(a) Sender (192.168.1.1)

```
# ./qemu-system-x86_64
 -drive file=/nfs/debian-kvm001.qcow2,if=virtio
 -boot d -enable-kvm -monitor stdio -vnc :0
 -incoming kemari:tcp:0:4444
# kill -256 (pid)
```

(b) Receiver (192.168.1.2)

Figure 2: Demonstration of INTERCEPT system



(a) 20MB Snapshot (b) 50 MB Snapsthot (c) 100 MB Snapshot (d) 500 MB Snapshot

Figure 3: The average turnaround time in the cases of Incremental Live Brock Migration(ILBM), Live Memory Migration with AUFS (LMM+AUFS), and Kemari-based Live Memory Migration with AUFS (INTERCEPT).

tion PM puts the base image file into `/root` directory, the source PM also puts the base image file into `/root`.

2. Sharing snapshot
   The source PM creates the snapshot and place the file into NFS server. The destination PM mounts the the NFS server as a read only file systems, and configure AUFS directory. For example, a NFS server exports `/nfs` directory and the destination PM mounts the place as a read-only file system, e.g, `/ronfs`. The PM then mounts `/nfs` as an AUFS system, in which `/ronfs` is a read only directory and any other directory, e.g., `/tmp` is a writable directory.

Instead of ILBM, this methodology employs copy-on-write to keep consistency of the VM disk image files between the source and destination VMs. We observed that the copy-on-write process will be started after the memory migration finished.

Finally, we employ Kemari [16] for reducing the time for memory migration. Kemari provides the feature of the fault tolerance for KVM, and makes the memory migration to be done in the background. The feature also enables the reduction of the time for migrating memory in INTERCEPT.

Therefore, our prototype implementation is developed by modifying the latest version of Kemari. The figure 2 demonstrates how INTERCEPT works for the sender and the receiver VMs. The sender (physical machine) runs the VM as shown in Figure 2a and the receiver also runs the VM as shown in Figure 2b with enabling Kemari's fault tolerant feature. The sender also starts the fault tolerant migration via sender's QEMU monitor console with specifying the IP address of the receiver. In the case of Kemari, the sender VM still runs after finishing migration; the receiver VM does not start, but is suspended. After the receiver VM starts, the sender disables the fault tolerant feature and stops the sender VM by calling `vm_stop(0)` from `migrate_ft_trans_error()` function in `migration.c`. As we mentioned above, we comment out the `vm_stop()` function not to power-off the VM. In addition, we also modify the receiver for remotely starting the VM. Instead of inputting command in the receiver's console, INTERCEPT accepts the signal for launching the VM, as shown in Figure 2b.

5. EVALUATION

The requirements of the INTERECEPT were *avoidance of data corruption* and *migration in short time period*. The section show the detail conditions of the preliminary evaluation in which we used two physical machines as shown in Table 2.

This evaluation employed three strategies for launching a honey server, namely Incremental Live Block Migration (ILBM), Live Memory Migration (LMM) with using AUFS for copying block devices, and INTERCEPT, our modified version of Kemari with AUFS. We also prepared four types of snapshot images, whose size were 20, 50, 100, and 500 MegaBytes (MB) in respectively. Note that 20 MB is the minimum size for the snapshot image file in our experiment.

Based on the above conditions, we measured the time for launching virtual machines. For the cases of LMM with AUFS and INTERCEPT, we compared the timestamp of the snapshot images created by copy-on-write with the time which started the migration. As for the case of ILBM, we manually measured the turnaround time by calculating from the started time and the completed time. The results are

summarized in Figure 3a, 3b, 3c, and 3d, where x axis denotes three cases, and y axis denotes the turnaround time while creating honeypot. Note that our x axis range for each box graph is limited to the 20 seconds, for readability.

When the disk size was 20 MB, we observed that the minimum average of the turnaround time was 2.00 seconds in the case of INTERCEPT, followed by LMM with AUFS (11.72) and finally ILBM (14.90). In order to compare the responses in a less biased way, we performed Analysis of Variance (ANOVA) and Welch's t-test ($p < 0.05$) for INTERCEPT and ILBM, and the result showed that there was statistical difference between the two turnaround times in between INTERCEPT and ILBM ($p \fallingdotseq 1.40E-32 (< 0.05), \nu = 18$). In addition to the INTERCEPT and LMM with AUFS, there also found statistical difference ($p \fallingdotseq 7.62E-33 (< 0.05), \nu = 18$).

Even if the disk size was 500 MB, the minimum average of the turnaround time was 43.23 seconds in the case of INTERCEPT, followed by LMM with AUFS (53.22) and finally ILBM (56.51). According to our ANOVA results, we performed Student's t-test ($p < 0.05$) for INTERCEPT and ILBM and found the statistical difference ($p \fallingdotseq 3.78E-16 (< 0.05), \nu = 11.86$). There was also statistical difference between INTERCEPT and LLM with AUFS ($p \fallingdotseq 4.00E-23 (< 0.05), \nu = 18$).

We also verified whether or not the data corruption occurred, and observed that there was no data corruption in all cases. Aspect from these observations, INTERCEPT succeeded to meet our requirements, *avoidance of data corruption* and *migration in short time period*. However, we also found that the turnaround time increases if the snapshot file size became bigger. We will discuss this problem in section 6.2.

6. DISCUSSION

6.1 Collection of attacks

Our motivation is to collect the information related to cyber attacks toward web applications using the honeypot. This section explains about the content of the information.

As shown in Figure 1, our concept contains detecting a suspicious request, creating a honey web server, preparing honey database, directing the request to the created server, and observing the behavior. Imagine if the request can be detected attacks without doubt. When the attack was well known and contained particular phrases that cause injection attacks, the rule-based detection can identify that the request is determinately attack. In this case, the information of the attack is not so new.

Aside from the well known attacks, we want to analyze suspicious attacks with honeypot. As we mentioned in section 1, heuristics-based methods calculate the likelihood of being a malicious code and compare the likelihood with the defined discrimination threshold. Assuming if the calculated score 0 means benign and 1 means attacks, and sore 0.5 is the threshold. For example, given calculated score 0.9 (> 0.5), it would be detected as attack rather than benign. However, even if the calculated score is 0.1, it might contain some suspiciousness. By observing the activity during the request, we considered that there is a potential chance for collecting new cyber threats.

Due to the fear of false positive, which is to label benign request as attack, a suspicious request might not be blocked. Instead, our approach seamlessly quarantines the suspicious request to the honey web server. Even if the false positive occurred, the service for the benign user of the web application would be continued without losing any convenience when following two conditions are met: (i) the honey web server has the same memory and disk information of the real servers, and (ii) the honey database equips the information that used by this benign user. If the honey database has limited tables and records that are related to the user who logged in to the web applications, as we explained in section 3, INTERCEPT might not penalize users' convenience even if the false positive error occurred.

6.2 Deception ability

This section explains about deception ability. We developed INTERCEPT to create a perfect copy of environment for web applications based on virtualization techniques. To the best of our knowledge, it would be difficult that honey web server has the same memory of the real web server, including TCP session state and web application session information, without using VM.

The remain issue is time for creating the copy of VMs. If it requires a lot of times, remote attackers will be aware of unusual or strange behavior of the web servers.

Our INTERCEPT was designed to reduce the time for migration; use of Kemari and its fault tolerant feature to reduce the time for memory migration, and use of AUFS and its copy-on-write feature for live block migration. The rest of time is the time for copy-on-write in AUFS. The creation time increases when the size of the snapshot disk image file becomes larger. The periodical "re-basing", the procedure of merging the base and the snapshot, might be necessary to keep the size of the snapshot to be reasonable.

The other solution is that use of rapid migration techniques. INTERCEPT needs replicate the VM instance rather than migration, so we need to carefully choose the suitable techniques to avoid data corruption. As well as as Kemari's fault tolerant functions, which do memory migration in the background manner, the fault tolerant functions for block disk image might be feasible. Actually, cloud storage techniques such as GlusterFS [9] and/or Sheepdog [23] have the fault tolerant features and replicate the disk images in the storage network. By modifying their replication functions, there is possibility for creating the perfect copy of VMs in very short time period even if the disk size is large.

Toward development of honeypot for web applications, INTERCEPT needs to interconnect to other modules, namely determination modules for suspicious requests and direction modules for them. On the integration of INTERCEPT and these modules, it needs to define the temporal requirements. The required time for detection, direction and preparation of honeypot might be calculated along with each requirement, but it is beyond the scope of this paper.

7. CONCLUSION

The paper introduced INTERCEPT, the core component of the honeypot for web applications. To meet the requirements of the honeypot, we explored the suitable solution and discovered the missing piece, which can avoid the data corruption as well as finishing the migration in short time period. While we employed server-type and high-interaction as a honeypot architecture, the honeypot system created the perfect copy of the actual system. Hence the honeypot had

the same memory and disk content of the actual system, web attackers would not be aware of that they were directed to honeypot even they checked any TCP and/or web sessions.

We also surveyed several solutions of virtual machine environments and developed our own virtual machine systems. At first, we chose QEMU and modified its source codes, and used the incremental live block migration for creating honey web server. Next, we employed AUFS instead of using incremental live block migration in order to reduce the time for completing live block migration. Finally, we modified Kemari and its fault tolerance feature, to reduce the time of the memory migration. We also observed that the creation of the honey web server could be done in few seconds when the size of the snapshot was reasonable.

The rest of work is to improve the performance of our developed INTERCEPT. We will also integrate the INTERCEPT into other modules that can detect suspicious requests and direct the requests to the honeypot systems. There still remains the problem, i.e., the difference of the temporal requirements, but we will develop the suitable honeypot for web applications regarding to the rapid migration techniques in our future work.

Acknowledgment

This research has been supported by the Strategic International Collaborative R&D Promotion Project of the Ministry of Internal Affairs and Communication, Japan, and by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 608533 (NECOMA). The opinions expressed in this paper are those of the authors and do not necessarily reflect the views of the Ministry of Internal Affairs and Communications, Japan, or of the European Commission.

8. REFERENCES

[1] Akiyama, M., Iwamura, M., Kawakoya, Y., Aoki, K., and Itoh, M. Design and Implementation of High Interaction Client Honeypot for Drive-by-Download Attacks. *IEICE Transactions 93*, B(5) (2010), 1131–1139.

[2] Andrews, M. Web Security 101 - introduction. Tech. rep., McAfee, 2008.

[3] Baecher, P., Koetter, M., Holz, T., and Dornseif, M. The Nepenthes Platform: An Efficient Approach to Collect Malware. In *Proceedings of the 9th International Symposium On Recent Advances In Intrusion Detection* (Sep 2006).

[4] Bellard, F. QEMU, a Fast and Portable Dynamic Translator. In *Proceedings of the USENIX Annual Technical Conference* (Apr 2005), pp. 41–46.

[5] Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., and Warfield, A. Live Migration of Virtual Machines. In *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation* (May 2005).

[6] Cohen, F. Deception Toolkit. Available at: `http://www.all.net/dtk/index.html`.

[7] Cook, S. A Web Developer's Guide to Cross Site Scripting. Tech. rep., The SANS Institute, Jan 2003.

[8] Dermann, M., Dziadzka, M., Hemkemeier, B., Hoffmann, A., Meisel, A., Rohr, M., and Schreiber, T. Best Practices: Use of Web Application Firewalls. Tech. rep., The Open Web Application Security Project, 2008.

[9] Gluster, Inc. GlusterFS. Available at: `http://www.gluster.org`.

[10] Kim, I. M. Using Web Application Firewall to detect and block common web application attacks. Tech. rep., The SANS Institute, Nov 2011.

[11] Lin, K., Kyaw, L., and Gyi, P. Hybrid Honeypot System for Network Security. *World Academy of Science, Engineering and Technology 24* (2008), 266–270.

[12] Mairh, A., Barik, D., Verma, K., and Jena, D. Honeypot in Network Security - A Survey. In *Proceedngs of the International Conference on Communication, Computing & Security* (Oct 2011), pp. 600–605.

[13] McDonald, S. SQL Injection: Modes of Attack, Defence, and Why It Matters. Tech. rep., The SANS Institute, Apr 2002.

[14] MITRE. Honeyclient Project.

[15] Nmap. Free security scanner for network exploration & security audits. Available at: `http://nmap.org/`.

[16] Ohmura, K., and Moriai, S. Kemari Project. Available at: `http://www.osrg.net/kemari/`.

[17] Okajima, J. R. Advanced multi layered unification filesystem. Available at: `http://aufs.sourceforge.net`.

[18] OWASP. OWASP Top 10 for 2013 - The Ten Most Critical Web Application Security Risks. Tech. rep., The Open Web Application Security Project, 2013.

[19] Pouget, F., and Holz, T. A Pointillist Approach for Comparing Honeypots. In *IEEE Conference on Detection of Intrusions and Malware & Vulnerability Assessment* (Jul 2005).

[20] Purcell, J. Web Based Attacks. Tech. rep., The SANS Institute, 2007.

[21] Saito, T. Proposal of DDoS attack mitigation using two-step map table lookup on LISP. Master's thesis, Nara Institute of Science and Technology, Feb 2013. (in Japanese).

[22] Seifert, C., Welch, I., and Komisarczuk, P. HoneyC-The Low-Interaction Client Honeypot. Tech. rep., Victoria University of Wellington, 2006.

[23] Sheepdog Project. Sheepdog Project. Available at: `http://sheepdog.github.io/sheepdog`.

[24] Spitzner, L. *Honeypots: Tracking Hackers*, 1st ed. Addison Wesley, 2002.

[25] Sundar, J. Livebackup - A Complete Solution for making Full and Incremental Disk Backups of Running VMs. Available at: `http://wiki.qemu.org/Features/Livebackup`.

[26] The Client Honeynet Project. Capture-HPC. Avaiable at: `http://client-honeynet.org/`.

[27] The Monkey-Spider project. Monkey-Spider. Availabel at: `http://monkeyspider.sourceforge.net/`.

[28] Wolf, K. block: drive-backup live backup command. Available at: `http://lists.nongnu.org/archive/html/qemu-devel/2013-06/msg04448.html`.

[29] Zalewski, M. p0f. Available at: `http://lcamtuf.coredump.cx/p0f.shtml`.