Honey-Copy - A Concept and Prototype of a Generic Honeypot System

Olivier Favre*, Bernhard Tellenbach*, Jan Alsenz†
*Zurich University of Applied Sciences, Switzerland
†Oneconsult AG, Switzerland
email: {favr,tebe}@zhaw.ch*, jan.alsenz@oneconsult.com†

***Abstract*—In this paper, we present Honey-Copy, a concept and prototype for a honeypot system that can pinpoint modifications caused by attacks or intrusion for any honeypot. To achieve this, we track modifications without having to install any additional tools on them. We make use of cloning to identify whether or not a modification has been caused by the honeypot itself or an attacker or intruder. We briefly present our initial prototype and discuss the challenges to be solved toward a more complete and feature rich version of our prototype.**

***Keywords*–Honeypot; Detection; Security; Monitoring;**

I. Introduction

Honeypots are decoy computer resources whose value lies in being probed, attacked or compromised [1]. The main difference between a normal computer resource and a honeypot is that the honeypot is not part of the production infrastructure [2]. One notable exception is the concept of Shadow Honeypots presented in [3]. As a consequence, attack detection methods do not have to cope with arbitrary production activity and the extraction of traces of attacks or intrusions is much simpler. After all, the traces do not submerge in production activities [2]. Honeypots are therefore a valuable tool to improve detection and reaction. However, since they do not protect a production infrastructure directly, they must be integrated with traditional security controls [4].

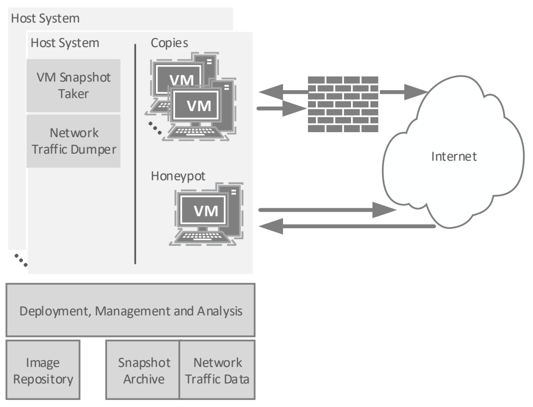
The lack of off-the-shelve products and solutions that allow automated and easy creation and monitoring of honeypots might be one of the reasons why the list of security controls used by a company does rarely contain one. Another reason might be that even though there exists many different kinds of honeypot systems and methodologies to analyze data produced by them, there is no system that satisfies all of the following four properties [2]: (1) the honeypots are not recognizable as such, (2) they are easy to configure and deploy, (3) the system reports activities related to attacks and intrusions only, (4) the core mechanisms (deployment, reporting of activities) work for any honeypot. Properties one and three are probably the most important ones. If these are not met, the system is of limited use since it would be easy to detect and it would be difficult to extract useful information from its reports. Properties two and four are relevant from an operational and business perspective. One of the major challenges is finding a solution to the problem of reporting activities related to attacks and intrusions without having to craft honeypot-specific algorithms or rules. At first glance, assuming that any activity on the honeypot is suspicious and should therefore be reported seems like a simple solution to this problem. After all, there is no production activity on a honeypot. While this often-made assumption might hold for activities like incoming network connections, it does not fit activities like the creation of a process or the modification of a file. Depending on the honeypot itself, we might see a significant amount of activity even on an ”idle” system. This includes things like automated software updates, an application-specific timed or event-based tasks (e.g., sync or cleanup tasks) or log entries from arbitrary scheduled tasks. Furthermore, when considering property four, the assumption about incoming network connections might be wrong too - a honeypot might do updates using active FTP or it might run a distributed service that sees incoming connections from other parts of the service from time to time. It is therefore crucial to have a generic way to distinguish between activities of type *self* and *third party* with the former including any activity triggered (or expected) by the honeypot itself.

In this paper, we present the main idea and concept of Honey-Copy, a system that should overcome most of the limitations of today’s honeypot systems. Our main contribution is a generic method to distinguish between activities of type *self* and *third party* and its integration in a general concept for a honeypot system. First, we provide an overview of Honey-Copy and discuss the basic idea of our generic approach to identify whether or not an activity is triggered by the honeypot itself or a third party (Section II) and we explain why an implementation of such a system is likely to be limited to high-interaction honeypots. Next, we introduce our prototype and discuss its implementation and evaluation (Section III). We then conclude our paper with a section on the challenges and next steps toward a more advanced version of our prototype (Section IV). A discussion of relevant related work can be found at the end of the paper (Section V).

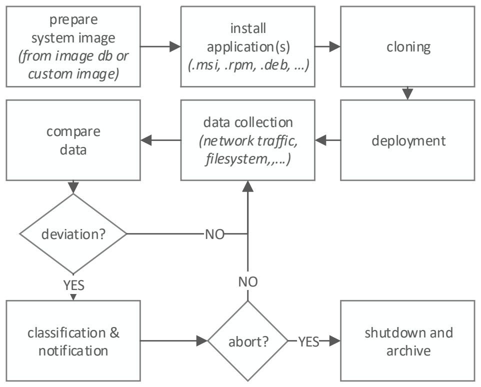
II. Honey-Copy - Basic Concept

The core idea of Honey-Copy is to make a clone of a honeypot and to put it behind a firewall that blocks incoming network connections. Since the clone cannot be accessed by third-parties, it exhibits activities of type *self* only. It should therefore be possible to identify and filter those activities from the reports of the honeypot system exposed to the attackers. Unfortunately, things like applications that create (temporary) files with random filenames or software updates that happen at different points in time make it difficult to implement an accurate and timely matching. This is why Honey-Copy can make up to n clones of a honeypot. By comparing them, patterns like random filenames can be identified and the chance that honeypot and clone(s) exhibit a certain activity of type *self* within a short time span can be increased. Hence, it should be possible to satisfy property (3).

Figure 1a shows the basic building blocks and setup of Honey-Copy. It consists of physical machines that make use of virtualization to run a host system and potentially many guest systems. The use of virtualization enables Honey-Copy to clone and deploy anything that can be provided in the form of a virtual machine image. This meets property (4) with respect to deployment. But this is not the only benefit of



(a)


prepare system image (from image db or custom image)
install application(s) (.msi, .rpm, .deb, ...)
cloning
compare data
data collection (network traffic, filesystem,,...)
deployment
deviation?
NO
YES
NO
classification & notification
abort?
YES
shutdown and archive

(b)

Figure 1. (a) Shows the basic setup of the Honey-Copy system and (b) the basic procedure it uses to deploy and manage a honeypot.

virtualization. Its use enables us to do the tracking of activities in the guest systems without installing any additional software and change to their configuration. In contrast to other honeypot systems, the honeypots deployed can be virtually identical to the production system they pose as. Hence, Honey-Copy can be said to satisfy property (1); the honeypot system itself does not make its honeypots more recognizable as such by itself.

The other two components running on the host system are the *VM Snapshot Taker* and the *Network Traffic Dumper*. The former can take snapshots of the guest systems, for example file-system or memory snapshots. The later can dump information about their network activity, for example full packet traces or flow level information only. This data can be captured and added to the *Snapshot Archive* or the *Network Traffic Data Archive*. It is then analyzed by the *Deployment, Management and Analysis* component, the heart of Honey-Copy. With the data available about the systems, different kinds of information like file or registry changes, running processes or network traffic can be checked for changes. From a data analysis point of view, the problem of identifying activities of type *self* by comparing this data from the firewalled clones and the honeypot itself is largely independent of the actual honeypot. The component has to learn or identify activities of type *self* from the clones and filter them from those reported by the honeypot. Hence, since the deployment mechanism as well as the activity-reporting works for any honeypot, Honey-Copy can be said to satisfy property (4).

In addition to the analysis task, it is also responsible for managing an Image Repository and for customizing and deploying images upon request. Unfortunately, it seems difficult to implement this in a generic and easy-to-use way for arbitrary honeypot types and configurations. One option to satisfy property (2) is to implement Honey-Copy for high-interaction honeypots only. In contrast to low- and medium interaction honeypots, these are real systems and not (partially) emulated or simulated ones. It seems, for example, practical to create a repository of images for many different operating systems and to use their software packaging and configuration mechanisms to quickly make and deploy a system that is a copy of a production web server or any other server or computer in a company network. In a corporate environment it would also be possible to use production server templates and mechanisms as a basis for this process.

Figure 1b illustrates the basic procedure to setup and manage an arbitrary high-interaction server honeypot in the Honey-Copy system. The first step is to choose an image from the *Image Repository* or to provide a custom image via an upload function. Next, additional applications and services can be installed and configured by providing them as package in the packaging format of the operating system, for example .msi for Windows or .deb for Debian Linux images or by using the configuration or packaging tool used by the organization. The packaging must allow for a fully automated installation of the application or service. Now that the image is ready, it can be cloned n-times and the honeypot and its clones can be deployed on the same or multiple physical hosts as outlined in Figure 1a. Data collection starts at the same time as the honeypot and its clones are turned on and does not stop until this honeypot is shutdown. In regular intervals, the data collected by the clones and the honeypot is compared and if deviations are found, they are classified and notifications are sent to those that subscribed to them. It is then checked whether the deviation found requires taking the honeypot offline, for example because it was hacked and the intruder started to attack 3rd parties in the Internet.

III. Prototype

In theory, the Honey-Copy concept meets all of the four desired properties. However, it is unclear whether or not it can be put into practice. To understand the related problems and challenges better, we built a an initial prototype of Honey-Copy.

A. Implementation

Our prototype consists of some Python scripts and a set of tools orchestrated by them. To manage and deploy the honeypots and their clones, we make use of Vagrant [5], a tool that is often used to create and configure lightweight, reproducible, and portable development environments. To deploy a honeypot, we first create a configuration file that specifies the type of machine to be used, the software that needs to be installed, and the way to access it. Based on this file, Vagrant can then create, deploy and launch an image for VirtualBox [6], a hypervisor that integrates well with Vagrant. When doing so, our prototype makes sure that the honeypot is cloned and that data capturing and the processes to detect activities other than *self* are in place and started.

For now, data capturing consists of recording full packet traces with tcpdump and snapshots of the file systems every *T=3600* seconds. This interval of one hour was chosen mainly to investigate the longer-term deviations between the clones and provide examples for activities of type *self*. For an actual detection setup, a much smaller interval is expected to be put in place. Whenever a new set of snapshots has been taken, the file systems of the honeypot and the clones are reconstructed, mounted and then scanned for differences using rsync. The reconstruction is required because we take differential snapshots to save storage space. In parallel to the file-system analysis, Pyshark with custom filters and rules (IP-Addresses, DNS-Names) is used to scan the network traffic dumps to extract communication partners that have not been seen by the clones. The result of the detection process is a report consisting of the differences in the file systems and the communication parters that are unique to the honeypot.

B. Evaluation

For an initial evaluation of our prototype, we deployed and tested the system with Linux and Windows based clones of typical web servers. For the evaluation, we compare the current status of the system to a perfect implementation of the Honey-Copy concept in terms of stealthiness, ease of deployment, attack detection and generic core mechanisms:

Stealthiness: The only two limitations of the prototype are that taking a file system snapshot of a virtual machine requires to suspended it and that the honeypots and clones are not physical but virtual machines. An attacker could detect the former using well-timed queries to the machine and the later might be achieved using fingerprinting methods like [7]. But the virtualization solution also supports snapshots of running machines, which could be implemented to mitigate the first limitation and as most organizations are using virtual machines at least in parts of their production infrastructure, this fact cannot be used as a sole indicator for a honeypot. Additionally, the only trace of Honey-Copy in the guest system is Puppet, an open-source software configuration management tool for Windows and Unix-like systems which is installed on them by Vagrants provisioning system. However, unlike honeypot specific logging and monitoring tools, its presence is not telling very much and it can be easily replaced with other tools. Other limitations exist but they are not introduced by the prototype itself but depend on how the system and its environment is configured and operated. For example, a public hostname like honeypot.company.com could be suspicious when used for a web server. And a system running a discussion forum with no activity in it might also look suspicious. As these factors are outside of the control of our solution and can be highly application specific, we consider them as out of scope for the prototype.

Ease of deployment: The prototype comes with the basic mechanisms and capabilities required to implement a user-friendly and easy-to-use interface to configure and deploy honeypots. However, for now it, the only interface is a command line interface. Furthermore, the *Image Repository* contains a few base images only.

Attack detection: The current mechanisms used for filtering activities of type *self* produces a significant number of false positives. One reason for this is that the prototype compares the file systems of the honeypot and its clones using the most recent snapshot only. For example, we observed many false positives because of automated software updates that did not happen or finish within the same snapshot interval. Another reason is that the comparison uses exact file matching. This turns files that are semantically the same but that have a different filename (e.g., temporary files with random filenames) or content (e.g., logfiles) into false positives. Another limitation is linked to the report generated from comparing the network traffic to the honeypot and the clones. This report lists communication partners seen by the honeypot but not its clones. Unfortunately, it contains a lot of entries that are not really interesting. This includes for example legitimate partners like search engine bots or Shodan [8] or illegitimate ones doing reconnaissance using known methods and tools. Furthermore, because the comparison of communication partners is done using exact matching, it cannot cope well with endpoints like content distribution networks.

Generic core mechanisms: Management and deployment works with any honeypot that is based on Windows or a Unix-like systems since these are the systems supported by Vagrant/VirtualBox. The same is true for the data capturing and comparison mechanisms since it does not depend on the actual system run in VirtualBox. Note that Unix-like includes most Linux distributions, Android and Mac OS.

IV. Challenges and Next Steps

In summary, we can identify two main challenges that the next version of our Honey-Copy must address. First, the system must provide a user-friendly and easy-to-use interface to configure and deploy honeypots. This can be done by writing an graphical user interface that compiles settings like the base images and the applications to be used by the honeypot into a suitable Vagrant file. The second challenge is more difficult to address. The mechanisms to identify files that are not modified by activities of the honeypot itself have to be able to detect files that are identical from a semantic point of view but that differ in content and/or have a different file name. To achieve this, generic heuristics that can detect patterns in file names or in the content of the files could be used. Another option would be to employ machine learning to search for such patterns. Furthermore, to cope for changes that might happen at different points in time on the honeypot and the clones, the mechanism must consider multiple snapshots from different points in time. What this means in terms of a delayed reporting and alerting is an important point of the evaluation of such an approach. While the focus is clearly on the file system part, there is also room for improvement with respect to the communication partners (attackers) reported by Honey-Copy. Endpoints like Windows update servers should not be reported as problematic because the honeypot and the clones use a different server for the update. The main challenge here is to make the matching mechanism aware of content distribution networks and similar behavior, for example by using third party tools, domain name resolution analysis or URL based heuristics to detect them.

We plan to address these challenges in the next version of our prototype.

When these have been addressed, there are still many more ways that the Honey-Copy prototype could be improved. If we consider that activities of category *third party* can be subdivided further into *benign*, *attack* and *intrusion*, it becomes clear that depending on the purpose of the honeypot, it could be required that Honey-Copy can filter activities of type *benign* and maybe even *attack*. *benign* activities are triggered accidentally or without malicious intent. This includes scanning from legitimate sources like Shodan HQ [8] or search engine bots, connection attempts that are the result of someone mistyping an IP address or URL and backscatter [9] traffic. Activities of type *attack* are triggered by an attempt to compromise the honeypot, for example using the Metasploit framework [10] and those of type *intrusion* are triggered by a successful compromise of the honeypot. To identify them, it could be useful to correlate network and file system activities and to employ an intrusion detection systems like Snort or Bro to fingerprint known attacks. We plan to research whether and how this could be done without having to sacrifice the generic nature of Honey-Copy when moving toward the third version of our prototype. Any other improvements like for example the addition of memory snapshots to the sources of information, is left to prototypes beyond version three.

V. Related Work

High-interaction honeypot systems that have similar goals in terms of stealthiness, attack detection, ease of deployment and honeypot configurations (operating system, applications etc.) are HI-HAT [11], HoneyBow [12], and Sebek [13]. Like Honey-Copy, these systems are server honeypot systems. In addition, we review a number of projects in the client honeypot sphere that are interesting because of the way they approach the problem of differentiating between real attacks and other activities.

HI-HAT [11] implements a system which converts normal PHP web applications into usable server honeypots. Their solution mainly consist of two components: The first component converts an arbitrary PHP application into a honeypot by adding monitoring capabilities to functions that handle requests from the outside. The second component consists of a GUI which lets an operator analyze the data gathered by the honeypot. To decrease the amount of false positives (generated by web crawler or other legitimate requests) the system makes use of white- and blacklisting based on general attack patterns. Furthermore, it allows the creation of custom filters to take into account different behavior of applications. Similar to Honey-Copy, it implements a way to deal with false positives generated by generic PHP applications.

HoneyBow [12] on the other hand is a high-interaction server honeypot which is designed for generic applications. It makes use of virtual honeypots to automate the management and monitoring of the system. In order to collect the necessary data to detect an attack, it implements three different tools (MwWatcher, MwFetcher, MwHunter) that search for malware binaries in the virtual filesystem and the network flow. Similar to the methods used in Honey-Copy, the MwFetcher component compares the content of the honeypot filesystem to the one of a clean copy that was taken at the start of an operation. The files which were new or altered and are flagged as executables are then further processed as malware. MwWatcher on the other hand is installed on the honeypot itself and can detect changes to the filesystem in real time. MwHunter finally inspects the network traffic for packages that contain executable malware. Each of the tools used in Honeybow has its own advantages and limitations. The MwWatcher component for example can be easily detected and disabled by an attacker. While this approach increases the chance of detecting an attack it also decreases the number of attacks since the system can be easily identified as a honeypot. Furthermore, it cannot deal with updates on the honeypot since that would likely change a number of executables compared to the clean copy.

Sebek [13] is another popular high interaction server honeypot system. It provides a data capture tool which monitors all actions of an intruder by capturing all `sys_read` calls. Furthermore, it tries to capture and send the logged data as stealthy as possible. Nevertheless, there have been a number of publications, notably [14], which show a way to detect and even disable Sebek. Another limitation of the tool is that there is no filter for the captured data. A manual analysis is required to distinguish a real attack from a false positive caused by normal system activity.

In the area of client honeypots Capture-HPC [15] and Capture Bat [16] present similar ideas. Both systems are high-interaction honeypots that make use of exclusion and inclusion lists. The former specifies acceptable non-suspicious activities to be ignored by the detection mechanisms. The later contains activities that are considered to be malicious. Such lists can be created for resources like the Windows registry, the file system, or processes. Capture-HPC also supports regular expressions to group a number of exemptions together. Currently, these list have to be created by hand and both systems run on Windows only. Another limitation of the approach is that any change to the honeypot (software updates, different mix of applications, etc.) is likely to require a modification of the exclusion list. UW-Spycrawler [17] on the other hand, makes use of trigger conditions (blacklists) which are specific to the browser used in the client honeypot setup. These conditions define activities which cannot be caused by the browser itself. Similar to the use of whitelists, these lists have to be created manually for a specific application (browser). Shelia [18] on the other hand takes a different approach to the problem without white- or blacklisting. The researcher behind the project proposes a system where the focus is on a reduction of false positives. It gathers data of an attack by monitoring the registry changes and file system actions generated by a process. The detection of said attack is done by analyzing from which memory address an API call was invoked. Once this address is obtained, it is checked whether it points to an executable memory location. If this is not the case an alarm is generated. This method allows to detect buffer and heap overflows that are exploited by an attacker. The downside of such a system is that it produces a higher number of false negatives since there are ways to circumvent the detection [19]. Pwnypot [20] take this idea even further by implementing more methods to detect arbitrary shellcode. It can detect ROP-Exploits and ASLR/DEP-Bypasses used by attackers.

VI. Conclusion

The main contribution of this paper is a concept and an initial prototype of Honey-Copy, a system that uses cloning to address the problem of distinguishing activities of the honeypot itself from those of attackers. We explain why and how our concept could be used to build a honeypot system that comes close to a perfect one in terms of stealthiness, ease of deployment, reporting of activities triggered by attackers and independence of the core mechanisms from the actual honeypots to be deployed. Other systems violate at least one of these properties. Our evaluation of the prototype shows that the basic mechanisms of our concept work and allow for a stealthy and generic implementation of the system. However, to satisfy all of the properties, the current method to compare the state of the clones to the state of the honeypot has to be replaced by a more sophisticated one and an easy-to-use graphical interface to configure and deploy a honeypot has to be developed.

REFERENCES

[1] L. Spitzner, "The Honeynet Project: trapping the hackers," IEEE Security Privacy, vol. 1, no. 2, Mar 2003, pp. 15–23.

[2] M. Nawrocki, M. Wählisch, T. C. Schmidt, C. Keil, and J. Schönfelder, "A Survey on Honeypot Software and Data Analysis," e-print arXiv:1608.06249, Aug. 2016.

[3] K. G. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E. Markatos, and A. D. Keromytis, "Detecting Targeted Attacks Using Shadow Honeypots," in Proceedings of the 14th Conference on USENIX Security Symposium - Volume 14, ser. SSYM'05. Berkeley, CA, USA: USENIX Association, 2005, pp. 9–9. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251398.1251407

[4] E. Vasilomanolakis, S. Karuppayah, P. Kikiras, and M. Mühlhäuser, "A Honeypot-driven Cyber Incident Monitor: Lessons Learned and Steps Ahead," in Proceedings of the 8th International Conference on Security of Information and Networks, ser. SIN '15. New York, NY, USA: ACM, 2015, pp. 158–164. [Online]. Available: http://doi.acm.org/10.1145/2799979.2799999

[5] "Vagrant," HashiCorp, URL: https://www.vagrantup.com/ [accessed: 2017-02-14].

[6] "VirtualBox," Oracle, URL: https://www.virtualbox.org/ [accessed: 2017-02-14].

[7] C. Jämthagen, M. Hell, and B. Smeets, "A Technique for Remote Detection of Certain Virtual Machine Monitors," in Proceedings of the Third International Conference on Trusted Systems, ser. INTRUST'11. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 129–137. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-32298-3_9

[8] J. C. Matherly, "SHODAN the computer search engine," URL: http://www.shodanhq.com [accessed: 2017-01-30].

[9] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage, "Inferring Internet Denial-of-service Activity," ACM Trans. Comput. Syst., vol. 24, no. 2, May 2006, pp. 115–139. [Online]. Available: http://doi.acm.org/10.1145/1132026.1132027

[10] "Metasploit," Rapid7, URL: https://www.metasploit.com/ [accessed: 2017-01-30].

[11] M. Mueter, F. Freiling, T. Holz, and J. Matthews, "High Interaction Honeypot Analysis Tool," URL: https://sourceforge.net/projects/hihat/ [accessed: 2017-02-14].

[12] J. Zhuge, T. Holz, X. Han, C. Song, and W. Zou, "Collecting Autonomous Spreading Malware Using High-interaction Honeypots," in Proceedings of the 9th International Conference on Information and Communications Security, ser. ICICS'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 438–451. [Online]. Available: http://dl.acm.org/citation.cfm?id=1785001.1785045

[13] "Know Your Enemy: Sebek, A kernel based data capture tool," The Honeynet Project, Last Modified: 17. November 2003, URL: http://old.honeynet.org/papers/sebek.pdf [accessed: 2017-02-14].

[14] M. Dornseif, T. Holz, and C. N. Klein, "NoSEBrEaK - attacking honeynets," in Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004., June 2004, pp. 123–129.

[15] M. Puttaroo, P. Komisarczuk, and R. C. de Amorim, "Challenges in Developing Capture-HPC Exclusion Lists," in Proceedings of the 7th International Conference on Security of Information and Networks, ser. SIN '14. New York, NY, USA: ACM, 2014, pp. 334:334–334:338. [Online]. Available: http://doi.acm.org/10.1145/2659651.2659717

[16] C. Seifert, "Capture-bat download page," URL: https://www.honeynet.org/node/315 [accessed: 2017-02-14].

[17] A. Moshchuk, T. Bragin, S. D. Gribble, and H. M. Levy, "A Crawler-based Study of Spyware in the Web," in Proceedings of the Network and Distributed System Security Symposium, NDSS 2006, San Diego, California, USA. The Internet Society, 2006.

[18] J. R. Rocaspana, G. Portokalidis, P. Homburg, and H. Bos, "Shelia: a client-side honeypot for attack detection," 2009, URL: http://www.cs.vu.nl/˜herbertb/misc/shelia/ [accessed: 2017-02-14].

[19] J. Butler, "Bypassing 3rd Party Windows Buffer Overflow Protection," URL: http://phrack.org/issues/62/5.html [accessed: 2017-02-14].

[20] S. Jalayeri and T. Jarmuzek, "PwnyPot, High Interaction Client Honeypot," URL: https://github.com/shjalayeri/pwnypot [accessed: 2017-02-14].