A Security Mechanism for Web Servers Based on Deception

Constantine Katsinis[1] and Brijesh Kumar[2]
[1]Computing and Security Technology, Goodwin College, Drexel University, Philadelphia, PA, USA
[2]Security Research Group, Rapidsoft Systems, Inc., Princeton, NJ, USA

Abstract - *The use of deception to deal with an adversary has been a tool for military strategists, intelligence agencies and law enforcement authorities for a long time. In computer security, deception includes actions taken to deliberately mislead attackers and to thereby cause them to take (or not take) specific actions that aid in the defense of a computer system. In recent years, honeypots have helped strengthen computer security through basic deception, by using them to study various attacks and monitor the way in which they were being accessed by an intruder. However, despite the importance of deception in computer security, deploying deception in modern web servers and web-based applications has not been extensively studied beyond deploying honeypots. In this paper, we examine the use of deception in the case where an intruder attacks a web server as a first step of an intrusion designed to access data sources on an internal network. We examine the development of a deception module which can be hooked into the Apache web server to detect malicious use of scripts and provide a deceptive response as necessary.*

Keywords: Deception, Intrusion Detection, Intrusion Response, Information Security, Information Warfare.

1 Introduction

Web services are becoming more ubiquitous and complex, making it more difficult to consider all possible exceptions of the proper behavior of web servers and applications, and leading to important security problems. Attacks against web applications constitute more than 60% of the total attack attempts observed on the Internet. Web application vulnerabilities such as SQL injection and Cross-Site Scripting flaws in open-source as well as custom-built applications account for more than 80% of the vulnerabilities being discovered [1]. However, an attack on a web servers is often only the first step of an intrusion designed to access data sources on the internal network [2,3].

Intranet websites can be attacked from the outside indirectly by first attacking a web browser on an internal network. Because corporate users sit behind firewalls they often have the access that is needed to attack intranet applications on behalf of an attacker. A victim visits a malicious web page, which assumes control of the victim's web browser. Subsequently, the victim's web browser can be instructed to make connections to servers on the internal IP range on behalf of the attacker. Essentially, the web browser of a user on an internal network becomes a launch platform for attacks against internal targets.

In this paper we focus on a similar case, where an intrusion starts with an attack on the front, internet-facing web servers of the enterprise. Because they cannot be relied upon to be completely secure, web servers are commonly located in demilitarized zones where they communicate with data sources on protected networks. In a DMZ, web servers are protected by IP/port-based filtering and can connect to machines on the Internet as well as other servers on the internal network which are usually less hardened due to cost/benefit tradeoffs. If an attacker gains control over the web server, then he can execute commands on these servers the same way that anyone at the keyboard of the server could. Initiating an effective attack requires interactive terminal access which can be accomplished by copying the shell interpreter to a folder within the document root of the web server. Several exploitation techniques exist for this purpose, including exploiting URL parsing, SQL injection, or targeting applications that validate input parameters poorly. Subsequently, the shell interpreter can be invoked by a URL to gather information about the internal network and attempt to escalate the attacker's privileges for the internal network.

Potential ways to reduce the effect of the attack may include prohibiting the web server from accessing the data source with the data source frequently updating the web server, or segmenting the network, placing the data source on a separate segment and limiting communications to the rest of the network. However, these measures do not eliminate the risks of an attack.

2 Deceiving the attacker

When an intrusion is detected, there are three options on how we might respond: 1) we may block the intrusion,

2) we may counterattack, trying to identify the adversary and neutralize his ability to attack, or 3) we pretend to be affected by the attack, leading the adversary to conclude that he has been successful, when in fact he is not.

Attempting to block the intrusion is the current standard practice, but the attacker quickly realizes that his intrusion is failing. In addition, although critical to network defense, access controls based on authentication and authorization can be defeated or circumvented. As a result, secondary lines of defense are also critical once access controls have been breached. Intrusion-detection systems, computer forensics, and honeypots are secondary lines of defense, but they are relatively passive and focused on data collection. Closing connections, ports, and services automatically during attacks stops them, but signals the attacker we recognize the attack and encourages a different attack.

Cyberattacks have been considered extensively within the context of large scale cyberconflict [4]. A general conclusion is that, although possible, a cyberattack (or counter-cyberattack in the subject of this paper) is not necessarily the best response to the intrusion; one reason is that unless it is completely successful, the attacker will immediately realize that his attack is failing and will switch to a different mode of attack [5,6]. An even more fundamental difficulty lies in the fact that cyber-attacks take place in an environment where attackers are connected to neutral third parties; a counterattack on a legitimate target may unavoidably damage a neutral party. This possible unpredictable damage, as well as weak attribution (identities are easily concealed or fabricated in cyberspace) reduce the effectiveness of counterattacks in cyberspace. Even worse, deterrence in cyberspace becomes more difficult since we cannot threaten unknown attackers and it is counterproductive to threaten the wrong party.

Option 3 is based on deception and can be the most beneficial, if the deception is maintained successfully for the proper amount of time. In its simplest form, deception occurs defensively, for example when a server increases processing times pretending to succumb to a denial-of-service attack so the attacker goes away. In general, deception is an extension of the concepts of intrusion detection and intrusion prevention. Intrusion Detection Systems issue alerts when they identify a potential threat in traffic patterns or when they detect in system logs a deviation from what is considered normal activity. Intrusion Prevention Systems have the ability to respond automatically to threats, taking some specific action such as dropping packets, terminating a connection or blocking network traffic from one or more hosts that are suspected to be malicious.

2.1 Beyond honeypots

For a long time honeypots have been used to simulate some parts of or a full operating system in order to collect data and study specific types of attacks. They are intend to be probed, attacked, or compromised. By definition they are computers connected to a network that no one is supposed to use; any connection is the result of an error or an attack. Therefore, honeypots have no production value allowing reliable forensic analysis of data with fewer false positives [21]. For example, [21] emulates the presence of multiple hosts to deceive attackers scanning a victim network. In [22] a honeypot is described which is used to analyze the behavior of an attacker after he breaks into a machine. The authors make it easy for an attacker to break in: they use weak passwords for ssh user accounts. In [23] a low-interaction honeypot platform (emulating only the vulnerable parts of a relevant service) is described which is used to collect information on self-replicating malware. Similarly [24] uses honeypots to collect malware data. In [25] a low-interaction web-application honeypot is described that emulates vulnerabilities to gather data from attacks that target web applications. It is designed to appear vulnerable to the attacker by providing the proper reply to the attacker's requests.

2.2 Intrusion Deception

The deployment of honeypots provides invaluable information on the behavior of attackers which can be used to improve Intrusion Deception techniques, but Intrusion Deception goes far beyond the deployment of honeypots. While honeypots are computers that no one is supposed to use, Intrusion Deception applies to fully productive computer systems that an authorized group of users is supposed to use.

Intrusion Deception goes beyond the basic security paradigm of detecting and reacting to threats. It comes from the realization that the basic security paradigm allows the attacker to always be one step ahead and plan his attack well in advance of our reaction. It attempts to be proactive and take advantage of the attacker's mentality and weakness. The understanding of the typical mentality of an attacker, his desire to cause damage, the use of attack tools, the way he probes a system after the initial intrusion, all can be used to create effective defensive deceptions as generic scenarios of excuses rather than isolated actions which may not be sufficiently convincing.

Previous efforts in including deception have focused on creating web servers or sections of web sites with "secrets" that only a malicious intruder would be interested in [7]. Since legitimate users would not be looking for secrets, any access to the "secret" area is assumed to be by an intruder, and the web server reacts deceptively by behaving as if encountering network errors or file retrieval difficulties. The purpose of this type of deception is to mislead the attacker and make him waste resources and time under the assumption that the server is under a time-critical denial-of-service attack or an intrusion where the attacker is relying on an unexpected and short-lived attack.

The concept of Intrusion Deception covers a wide range of responses, including decoy systems and fake information. It enhances information systems enabling them to deceive attackers to prevent them from achieving their goals [7-11]. Since attackers rely on responses from computer systems, such deception can be very effective with minimal resources whether attacks are initiated by insiders or outsiders. Its initial goal is to gather information about the nature of the attack but, unlike a honeypot whose goal is to lure and study attackers, its real purpose is to confuse, misdirect and frustrate a malicious attacker, while at the same time it collects intelligence and forces the attacker to expose his sources and methods. It also employs techniques to force the attacker to perform actions that are detrimental to his purpose, such as forcing him to use communication protocols that make it easier for the Intrusion Deception system to achieve its purpose. Deception is useful even when we are very sure of an attack, first as a delaying tactic, and by diverting the attacker to honeynets at the proper time as determined by the detected intrusion, until at some point it may become safer to disconnect the attacker.

A framework for using intelligent software decoys to deceive hackers once they have infiltrated a system is described in [12]. The model consists of a security contract, which when violated triggers the generation of deceptive decoys by the software object. The goal of this deception is to convince the mobile agent into concluding that it has successfully infiltrated the system. The decoys described simply consist of a fake java object generated at run time with randomly permutated arguments. Most of these techniques have been compiled in a software package consisting of PERL scripts called the deception toolkit (DTK) [13].

In the Intrusion Deception architecture, we continuously monitor network and server activities for suspiciousness. As suspiciousness increases, we first provide minimum deceptive measures, and then increase their frequency and severity. Deception is used sparingly and consistently to keep the attacker fooled as long as possible, tying up his resources while reducing his chances of successful attack. Thus, Intrusion Deception is a defense mechanism implemented on a real web server that continues to provide correct information to legitimate users (and to benign requests of a potential attacker) and engages deception only when an attack is verified to be in progress. Its fundamental purpose is not to mount a counterattack but rather to give the illusion to the attacker that he is succeeding. Therefore, it is only an additional level of defense of an otherwise fully functional web server, increasing the difficulty of a successful attack. The question of whether deception is ethical or legal has been extensively examined in the past and most ethical theories allow for deception against serious harm [14,15]. We consider an intrusion into a web server to be a serious harm. Legal issues in several countries are presented in [16] including whether the defender has a duty to disconnect the system under attack (to retreat from the attack).

While the range of Intrusion Deception techniques is extensive, in this paper we focus on engaging intrusion deception techniques to prevent an attacker from gaining control of a web server and using it as an intermediate step to launch attacks into the internal network of an organization. We focus on the situation where despite network defenses, an attacker has penetrated the network and has managed to install a malicious piece of software on the web server.

3 A deception module on Apache

One way of protecting critical processes, such as the Apache web server, is to enclose them in software wrappers [17]. A software wrapper is basically a set of rules for detecting and responding to suspicious behavior. It allows the interaction between the critical process and the external client (who might be a potential attacker), but employs deception techniques when it detects an intrusion. In its simplest form, the software wrapper responds with fake error messages, keeps the attacker occupied or redirects his traffic to a honeypot.

In a concept quite similar to a software wrapper, Apache modules have been developed to defend the server against attacks. Two common ones are ModSecurity [18] and ModEvasive [19]. ModSecurity considers common types of attacks such as variable-length buffer injection, meta character injection, and SQL injection that pass unobstructed through common firewall configurations, and attempts to detect the attacks and block the related traffic. ModEvasive is an Apache module providing evasive action in the event of an HTTP DoS or DDoS attack. It detects certain events, such as frequently requesting the same page, and denies access to the corresponding IP address. It can be configured to coordinate with firewalls and routers to optimize its response and reduce the required bandwidth and processor utilization. Although their purpose is to defend the server against attacks, they both rely fundamentally on access controls, and neither of them attempts to use deception.

In this paper, and as a proof of concept, we discuss a simple implementation of a deception module as part of the content generator module architecture of the Apache web server. Apache is using a modular approach to process an HTTP request allowing a module to handle a particular task but ignore other aspects of the request that are not relevant. At its core is the content generator. Modules register content generators by defining a function referenced by a handler configured by directives in the configuration file httpd.conf.

A request goes through several phases before being processed by the content generator. These phases examine and change the headers of the request to verify access rules, map the request to a file or script, and determine the proper content generator. The logging phase takes place after the content generator has sent the response back.

In Apache new modules can be developed and inserted into any of the processing phases described above. A module defines a function and, through the proper hook, tells Apache to call the function at the appropriate processing phase.

Although Apache allows modules to hook functions in the phases before content generation (called metadata modules), in this paper we present a simple example of hooking a deception function as the very first part of the content generator that responds to CGI requests.

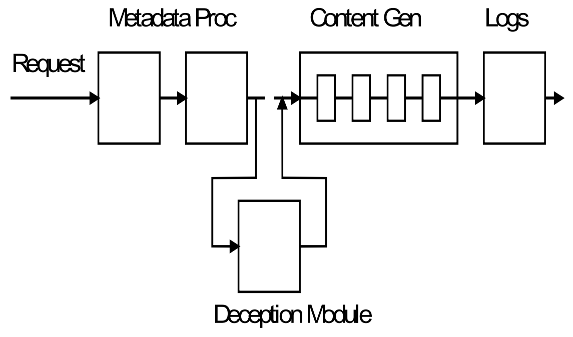

Figure 1. Deception module in Apache architecture

Figure 1 shows the architecture and location of the deception function, as well as other locations where more advanced deception architectures may include additional deception functions.

Extensive information on the design of Apache handlers is given in [20]. A summary of creating a hook is given below, where the basic data structure and necessary function calls are shown.

```
static int deception_handler
(request_rec *r) {
  /* if we are not interested,
     return DECLINE or ERROR */
  /* deception processing code */
  return DECLINED; /* or OK */
 }

static void deception_hooks
(apr_pool_t *pool) {
  ap_hook_handler(deception_handler,
   NULL,NULL,APR_HOOK_REALLY_FIRST);
 }

module AP_MODULE_DECLARE_DATA
deception_module = {
 STANDARD20_MODULE_STUFF,
 NULL, NULL, NULL, NULL, NULL,
deception_hooks
} ;
```

The module structure declares a function member to create a request processing hook. A hook also indicates which part of the request the module is interested in. Here we are interested in executing our function as early as possible in the content generator, and the ap_hook_handler() function is called with parameter APR_HOOK_REALLY_FIRST. Depending on the module requirements, a function may be hooked in different places withing the content generator, through the use of other constants in place of parameter APR_HOOK_REALLY_FIRST. These constants are defined in a header file in the Apache source, however other integers may be used within a proper range for finer control.

In general, the handler function (deception_handler) examines the data in the request data structure, decides whether to respond or ignore the request, and finally returns a code. Apache defines several return values: OK, indicating that the function has completely handled the request and no more processing is necessary; DECLINED, indicating that the function is not interested in this request and that Apache should call another handler; or a status code indicating an error.

The deception function examines the URL of the request and decides which steps to take next. In this paper we are interested in CGI scripts, so the function may return DECLINED very early if the current request is not for a CGI script. Otherwise, the function examines the script and may respond to the request and return OK to stop Apache from executing the script or it may simply return DECLINED to allow normal execution of the script. The ability to create deception lies in both choices: the deception function may completely respond to the request in several (possibly deceptive) ways with no need for further processing by Apache, or it may modify the environment in which the script may run and allow Apache to run it.

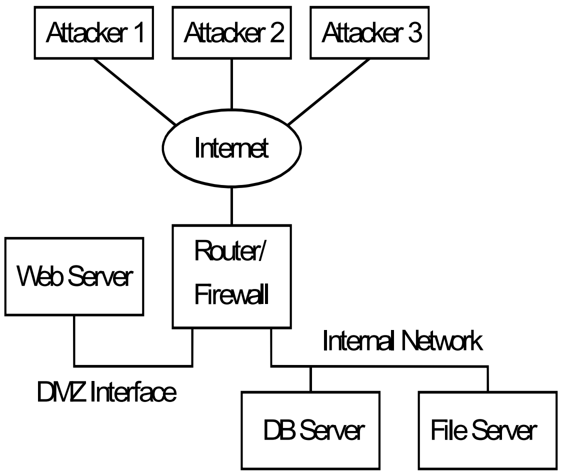

Figure 2. Experimental Setup

In this paper we use the experimental setup shown in Figure 2. We are not only interested in analyzing attacks in-depth, but also in gaining a general understanding about the effectiveness of deception scheme when attackers interact with the web application. To simulate the mounting of an attack we use three PCs running scripts to generate various Web attacks, including a) SQL injection attack, b) Brute force attack to break the password, and c) attempt to access files or folders with insufficient authorization. We assume that the initial intrusion has been successful and that the attacker has succeeded in either copying the shell interpreter or installing a similar program in a folder within the document area of the web server. Specifically, in our experiment we use Apache 2.2 running on Windows 7 and we assume the existence of a CGI program (written in C) that takes as a parameter one string with a DOS command, execute it and display the result back in the attacker's browser. This program is in the cgi-bin folder.

One of the first actions an attacker would take after installing this program is to examine the contents of folders on the server file structure that he would not normally have access to. Our deception function can examine the parameters of every script and in this particular case detect a suspicious DOS command. It has the choice of allowing the request to go through and be executed, if for example the target folder is under the DocumentRoot structure (and folder listing is allowed), or it may transform the string of the DOS command to a benign command and keep the history of requests that will maintain the deception in future requests by the attacker.

4 Use of log information by the deception module

Log information can greatly assist the deception function. The web server logs as well as system logs can reveal what is actually happening on the server and provide the history needed to create a successful deception. The fact that logs contain information on what has already happened may be seen as a drawback as they are not effective in blocking a certain response or shutting down the connection. However, the attacks considered in this paper consist of several steps over some period of time, during which we may have the opportunity to detect the suspicious activities and create a response based on deception.

Extracting information from the logs can be implemented as additional modules in Apache, with functions inserted before (metadata phase) and after (log phase) the content generation phase. These functions parse the data, extract patterns, maintain history and context, combine information from what may seem to be distinct attacks, decide on further deceptive actions, and produce their own logs.

The deception process can also benefit from more sophisticated anomaly-based pattern recognition systems which model normal (legitimate) traffic and intrusion attempts (anomalous) traffic. Such systems continuously collect a large sample of real requests on a web server and attempt to classify the requests based on the web server inputs, as attacks and legitimate requests, and possibly to identify different categories of attack.

Monitoring key system files also provides substantial information to the deception process, since most exploit tools use one or more system files. By monitoring the activity on such files in real time as well as the account under which the processes execute, the deception process may be able to substitute real data with deceptive data, rather than block the intrusion which would be immediately known to the attacker. These files include cmd.exe, ftp.exe and tftp.exe, ping.exe and net.exe.

5 Conclusion

It is becoming increasingly apparent that cyberdefense based on detecting and reacting to threats is not sufficient as it allows the attacker to always have the initiative. Counterattacks in cyberspace are also problematic as they carry the risk of damage to neutral third parties. Defense techniques based on deception can be beneficial if the deception is maintained successfully for the proper amount of time, leading the adversary to conclude that he has been successful, when in fact he is not. In this paper we examine the case of using deception to defend against an attack on a web server and further attacks on servers in internal networks and we describe the development of a module which can be hooked into the Apache web server to detect malicious use of scripts. The deception module examines the incoming HTTP request and can analyze the web server and system logs to decide if a deceptive response is necessary.

6 References

[1] SANS, "The Top Cyber Security Risks 2009", http://www.sans.org/top-cyber- security-risks/

[2] How Internal Network Becomes External, http://www.ddosed.com/uploads/ penetration_ testing/srgn-pentest-01.pdf

[3] Grossman, J., "Hacking Intranet Websites from the Outside Take 2", https://www. blackhat.com/ presentations/bh-usa-07/Grossman/Whitepaper/bh-usa-07-grossman-WP.pdf

[4] Lin, H., "Lifting the Veil on Cyber Offense", Security & Privacy, IEEE, July-Aug. 2009, v. 7 n. 4, pp. 15-21.

[5] Lewis, J. A., "The Korean Cyber-Attacks and Their Implications for Cyber-Conflict", Oct 23, 2009, CSIS Center for Strategic and International Studies.

[6] Owens, W.A., Dam, K.W., and Lin, H.S., editors, "Technology, Policy, Law, and Ethics Regarding U.S. Acquisition and Use of Cyber-attack Capabilities", Committee on Offensive Information Warfare, National Research Council, National Academies Press, 2009.

[7] Rowe, N.C., "A model of deception during cyber-attacks on information systems", Multi-Agent Security and Survivability, 2004 IEEE First Symposium on , pp. 21- 30, Aug. 2004

[8] Rowe, N.C., "Designing good deceptions in defense of information systems", Computer Security Applications Conference, 2004. 20th Annual , pp. 418- 427, 6-10 Dec. 2004.

[9] Rowe, N.C., Duong, B.T., Custy, E.J., "Fake Honeypots: A Defensive Tactic for Cyberspace", Information Assurance Workshop, IEEE, pp. 223-230, 21-23 June 2006.

[10] Rowe, N.C., "Finding Logically Consistent Resource-Deception Plans for Defense in Cyberspace", Advanced Information Networking and Applications Workshops, 2007, AINAW '07, 21st International Conference, vol.1, pp..563-568, 21-23 May 2007.

[11] Julian, M.D.P., Rowe, N.C., Michael, J.B., "Experiments with deceptive software responses to buffer-overflow attacks", Information Assurance Workshop, 2003. IEEE Systems, Man and Cybernetics Society, pp. 43- 44, 18-20 June 2003,

[12] Michael, J. B., Riehle, R. D., "Intelligent Software Decoys", Proc. Monterey Workshop on Engin. Automation for Software-Intensive Syst. Integration, ARO/ONR/NSF/DARPA, June 2001, pp. 178-187.

[13] Cohen, F., "The Use of Deception Techniques: Honeypots and Decoys", http://all.net/journal/ deception/Deception_Techniques_.pdf

[14] Bok, S., "Lying: Moral Choice in Public and Private Life", Pantheon, 1978.

[15] Nyberg, D., "The varnished truth: truth telling and deceiving in ordinary life", University of Chicago Press, 1993.

[16] Benichou, D., Lefranc, S., "Introduction to network self-defense: technical and judicial issues", Journal of Computer Virology, Springer, Vol.1, issue 1-2, pp. 24-32, Nov. 2005.

[17] Michael, J.B., "On the Response Policy of Software Decoys: Conducting Software-based Deception in the Cyber Battlespace", COMPSAC 2002, p 957-962

[18] http://www.modsecurity.org/

[19] http://www.zdziarski.com/blog/?page_id=442

[20] Kew, N., "The Apache Modules Book: Application Development with Apache", Pr Hall, 2007.

[21] http://www.honeyd.org/

[22] Alata, E., Nicomette, V., Kaâniche, M., Dacier, M., and Herrb, M., "Lessons learned from the deployment of a high-interaction honeypot", 6th European Dependable Computing Conference, Coimbra, Portugal, 18–20 October 2006, pp. 39-46.

[23] Baecher, P., Koetter, M., Holz, T., Dornseif, M., and Freiling, F.C. "The Nepenthes Platform: An Efficient Approach to Collect Malware", Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection, Hamburg, Germany, September 20-22, 2006, Springer 4219, pp. 165-184.

[24] Leita, C., Dacier M., "SGNET: A Worldwide Deployable Framework to Support the Analysis of Malware Threat Models", 7th European Dependable Computing Conf, Lithuania, May 2008, pp. 99-109.

[25] Rist L., Vetsch S., Kossin M., Mauer M., "Know Your Tools: Glastopf - A dynamic, low-interaction web application honeypot", http://honeynet.org/papers/ KYT_glastopf