A Framework for Intrusion Deception on Web Servers

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 - *Threats against computer systems continue to multiply, but existing security solutions that attempt to keep the attacker out of the system are becoming unable to keep pace with these challenges. In this paper we discuss the application of military deception to defend computer systems. Deception techniques enable the defender to influence the attacker's selection of targets and thus direct him to perform actions that reveal his presence and intentions. We discuss techniques that mislead attackers and cause them to take specific actions that aid in the defense of a computer system. We then focus on web servers, that are frequently attacked often as a first step of a deeper intrusion into a computer network, and present an architecture integrating deception into a popular web server.*

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

1 Introduction

Traditionally, security professionals have used two strategies for defending against attacks in cyberspace: identifying and fixing vulnerabilities and detecting attacks before they inflict significant damage. They have focused on keeping attackers from stealing data by actively watching for intrusions, strengthening perimeter defenses, blocking attacks with network technologies such as firewalls, protecting against malicious software with antivirus technology and relying on defensive signatures.

However, these techniques have their limits. In fact, the Internet provides attackers with a common knowledge base. They can research new vulnerabilities and find systems that exhibit these vulnerabilities. Given enough time and information an attacker can learn to circumvent a firewall. An IDS will only provide information after the attack has started, which often leaves little time to secure all vulnerable systems, in effect forcing administrators to wait until the damage is done. Then, their only option is to make defensive changes and relaunch as quickly as possible.

Recently, the thinking has evolved to the point where, while keeping the enemy out is paramount, it is assumed that attackers have gotten inside and will again. There is growing sense that organizations need to be more aggressive in fighting off intruders especially as the costs of digital espionage keep increasing.

Web servers in particular are exposed to the public and are easily examined by the outside world. An attacker can take the time to traverse the site, understand how web applications are coded and locate the defensive measures that are in place which allows them to avoid being profiled. This activity goes undetected because the server sees it as the traffic of a legitimate user. The attacker may continue the attack for example by locating dynamic pages, especially those which accept form or query input, derive boundary input cases and attempt to provoke an unintended response from the server. By repeating this process systematically, he obtains a list of all the parameters that are either properly validated by the server or not. He therefore obtains pages that are vulnerable because boundary values of their parameters produce calculation errors, fatal errors, or are injected into the response without cleansing. He then attempts to exploit these vulnerabilities by trying different attack vectors. Being detected or blocked at this point is not an issue as the attacker can continue the attack through another proxy.

Among the most important web application vulnerabilities, as they appear in the OWASP 2013 Top 10 project [17], are Cross-Site Scripting, Injection attacks and Path Traversal. Injection attacks include attacks based on input that may contain malicious code to be executed, such as the SQL injection attack. They also include Command injection, an attack that converts the application into a system shell that executes the attacker's commands on the operating system without even being logged on. If the web application runs under root permissions, then the attacker is able to run any possible command. Path Traversal attacks rely on the "insecure direct object reference" vulnerability which exposes a reference to a file or directory without proper access control check. The attacker can modify such a reference

to access a file outside the web root directory.

2 Intrusion Defense Evolution

Deception is invaluable in warfare and other conflicts as it can be used to trick an adversary into taking actions that waste his resources or move his resources to make them easier to attack. Deception-based defenses in information systems have been in use for a long time, from simple techniques such as the login process asking for a password even when the supplied user account does not exist, to the extensive use of honeypots and decoys since the early 1990s.

Intrusion Deception is an extension of intrusion detection and prevention with a primary purpose to confuse, misdirect and frustrate maliciously driven attackers. Early forms of Intrusion Deception techniques include spoofing service banners, labeling system services deceptively, routing threatening traffic to honeypot networks, integrating decoy systems within critical resources and placing tracking beacons on decoy files.

Intrusion Deception can create an environment where the attacker is uncertain if he has succeeded in intruding into the network and whether he has extracted the data he was searching for. Ultimately, though, the intruder should not even be aware of the deception. The goals of Intrusion Deception can be either to keep the attacker on the system in order to trace him or to make him leave. If the goal is to make him leave then he must be induced to lose interest or believe that he was successful in his attack. In either case, if he leaves on his own, he is unlikely to come back. In the end, Intrusion Deception costs more time to the attacker (in fruitless attacks and extraction of false information) than the defender and gives the defender information about the attacker's tools and motives to prevent the attack from causing damage.

Honeypots

Honeypots were among the first deployments of network deception. Honeypots appear to be a component of a larger network architecture. In reality, though, they do not contain any useful data and are often separated from regular network resources. They serve no purpose, except collecting data about attacks on them, and have no legitimate users and consequently can be deceptive all the time. However, they offer no utility after a successful attack has already occurred.

Low interaction honeypots like Honeyd [6] basically simulate the network protocols and respond to network probes. High interaction honeypots respond to network probes and permit logins and access to resources. Honeynets are groups of honeypots, imitating an actual or fictitious network, used to study how attacks spread from one computer to computer.

Honeypots can be recognized without deception, since attackers can see a lack of normal file structure and lack of temporary files. Several techniques are provided in [4] that an attacker could use to detect honeypots. There have been many proposals of deliberately deceptive activities on honeypot networks to keep attackers busy. Such activities include configuring the router to respond to many fake IP addresses, and augmenting the honeypot with a virtual storage system. These are a simple passive form of deception. In addition, honeypots do not follow an important principle of conventional warfare, that deception should be integrated with genuine operations. As argued in [14,15], deceptive tactics are more effective on real systems.

Typical honeypot deployments include:
a) a minefield, where honeypots exist among regular servers, possibly containing some of the real server data. An example deployment is placing honeypots among servers in the DMZ to capture attacks against the public servers and also servers in the internal network. This deployment can be effective against intruder stealth scanning ("slow scans") which may not set off an IDS system but will be detected by the honeypots.
b) a shield, where each regular server is paired with a honeypot deployed in a DMZ. A firewall redirects the network traffic according to the shielding policy: regular traffic is directed to the server while any suspicious traffic destined for the server is instead sent to the honeypot shield. The honeypot typically mirrors some noncritical content of the regular server to increase its deception.

Sticky Honeypot (LaBrea) is another deception technique used to protect networks and some applications. It takes over unused IP addresses, and creates virtual servers that are attractive to worms, hackers, and other denizens of the Internet. The program answers connection attempts in such a way that the machine at the other end gets "stuck", sometimes for a very long time [5].

3 Deception Strategies

Deception in computer systems relies heavily on the principles of deception in military conflict settings. Several types of military deception have direct application in computer systems, such as feints, lies, disinformation, ruses, concealment, camouflage, and manipulation of the adversary by insight into their reasoning and goals [3]. In a similar fashion, information systems can lie, cheat, and

mislead attackers to prevent them from succeeding [15]. Such deception can rely on minimal resources to be very effective in creating deceptive delays (to allow setting up a permanent defense to time-critical attacks) and defensive lies (to manipulate attackers).

Just as in military conflict, deception in computer systems is a way to foil attacks. Deception may convince an enemy to go away without any fight most probably to avoid an all out defeat. However, the ultimate success of deception in computer systems occurs when the enemy goes away thinking he has succeeded. A computer system can achieve this by relying on intrusion detection to monitor suspicious user behavior. As suspicious activity increases, the system increases its deceptive measures, keeping the attacker fooled as long as possible, tying up his resources, and allowing him to believe he is successful while reducing his chances of successful attack. Simple excuses can be combined to create deception: the file system has crashed, the required software has bugs, network parameters are incorrect, security policy is in effect, the attacker action caused the system to malfunction.

Planning is essential to deception because most often defensive lies and delays require consistency. The system needs to maintain, and be able to reason about, what has been presented to the attacker so far so it can decide what deceptive action to take next. For example, once network problems are used as an excuse to deny or delay some request, the same excuse (or similar evidence of network problems) should be given to following requests of the same type by the attacker. In other situations, an earlier excuse may not be sufficient for a subsequent refusal to a new request by the attacker. For well-defined situations resulting from known attacks a detailed plan may be constructed [10]. Otherwise alternative excuses may be ranked and selected [15]. [2] used attack graphs to guide the activities of attackers based on situation-dependent lies. The objective of the plan is to minimize damage to the system and can be characterized by the probability that the attacker goes away either having given up or thinking he has succeeded in his attack. Figure 1 shows the basic flow of engaging an attacker using a deception plan.

While planning of consistent deception actions is essential, inconsistent deceptions are also useful [13]. A consistent deception builds a fake reality that still functions under the rules of reality so the attacker does not see the deception. An inconsistent deception attempts to disorient the attacker. He will realize the inconsistency, even the deception, but will not know which of his observations relate to the real system and which to the deceptions. One possible outcome is that the attacker will leave without causing any damage to the system.

4 Decoys

Deployment of decoys in computer systems follows the basic idea of filling the attacker's search space with decoys so that detection of real targets becomes difficult. One common deployment of decoys is to defeat penetration testing tools used by attackers that probe target servers looking for vulnerabilities. Penetration testing tools identify operating system and server types and versions and provide facilities to perform attack sequences against identified vulnerabilities in the target system. However, these tools have characteristic behaviors which make them readily identifiable by the targets of their attacks. The defender can then simulate a variety of server characteristics and services (decoys) so that the attacker makes errors differentiating between real and fake targets, and is lead by the defender down defender-designed attack graphs that cause the attacker to waste resources or deceive him into thinking he is succeeding.

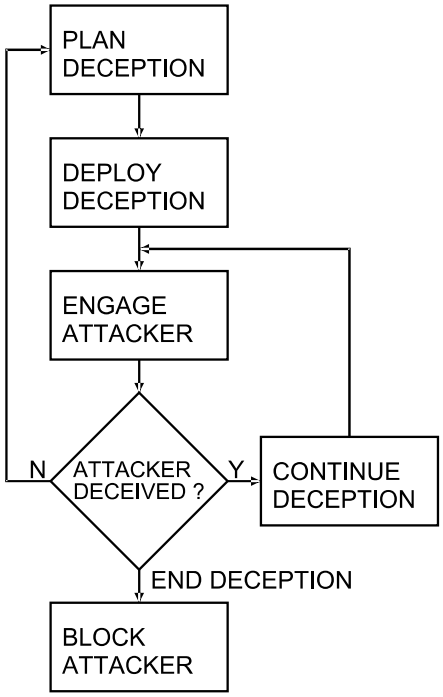

Figure 1. Deception plan deployment

In general, decoys are constructs which contain data that appears valuable but is in fact fake. They may be files, web page elements, fake data flows injected into a network, or fake but believable user activity on a server [1]. Figure 2 shows the basic network architecture. Decoys may contain random data or bait information which the attacker may attempt to use at a later time. They are effective because attackers lack the thorough

knowledge of the target system which authentic users have. Authentic users can distinguish or remember which resources are real and which are fake, and have no reason to access inauthentic decoys with no useful data, while an attacker will have difficulty differentiating decoys from desirable data. Decoys are able to defeat low quality attackers without interfering with normal users and can quickly indicate the presence of an attack. The target can respond by increasing security measures against the attacker including escalating its deception profile.

with name, content and attributes that are realistic; but it also must be variable, exhibiting as much variability as normal documents in the system, so that they are not easily detected by simple pattern-recognition processes. Decoys must be conspicuous and enticing so that they are located in places where the attacker will most probably search; but they also must avoid confusing the normal users or obstructing regular activities on the system. The operating system must be configured to make any decoy access detectable; use of decoy contents by the attacker

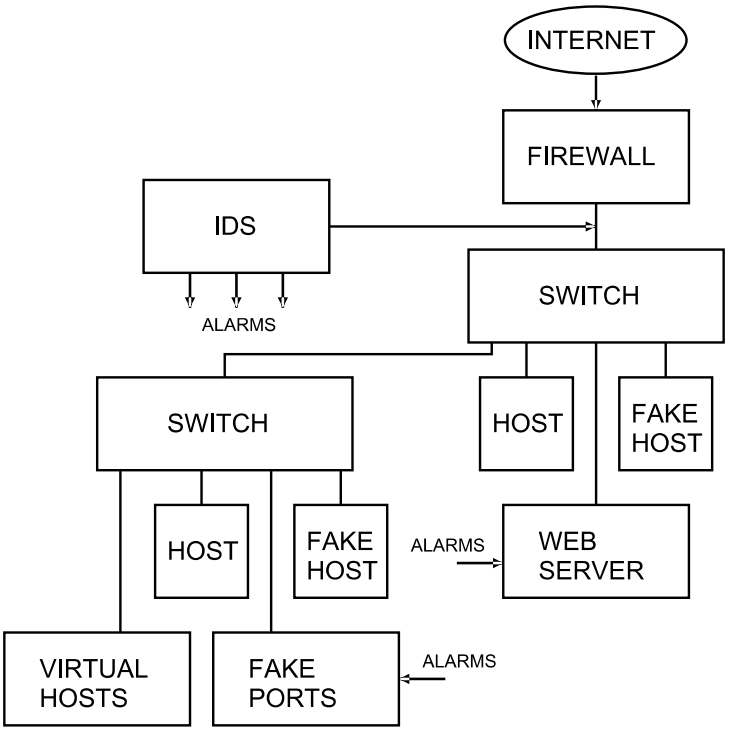

Figure 2. Deception architecture

Intelligent software decoys are special type of decoys or agents that protect objects from unauthorized access [11,12]. If an attacker attempts to access an object in a way that does not conform to the interface specification of the object, then the object changes its behavior from its normal operating mode to a deception mode. In deception mode the object attempts to deceive the attacker into concluding that its violation of the interface specification has been successful. In general, responses of such software decoys include maintaining the interaction with the attacker to learn about the nature of such interactions, terminating the interaction or even treating the attacker as a cybercombatant.

Effective decoys have certain characteristics [16] which enable them to maintain deception against attackers without affecting normal users. Decoys must be believable

can also serve as an alarm. Finally, decoys must be updated regularly to keep them believable, conspicuous and enticing.

5 The role of deception in Web attacks

There are many reasons why web servers are frequently attacked:

- Most of the application code (with vulnerabilities) is public on the website.
- The web server offers the quickest pathway to infiltrate the company network.
- The web server is largely undefended, reducing the possibility that the attacker will be detected.
- The skill level required to exploit known web server vulnerabilities is low because attack scripts are available

to download.

It is challenging to respond to Web application abuse because attacker probing is mostly invisible to intrusion detection systems, and cannot be easily distinguished from normal user behavior. Organizations maintain a layered defense in the form of network firewalls, intrusion protection systems and web application firewalls primarily based on signatures and anomaly detection. These security technologies are useful in blocking the known attacks but are not sufficient to prevent intrusions from unknown attacks for which no pattern exists.

While organizations can continue to rely on layered defenses, embedding deceptive technology into the web server can be used to block and misdirect attackers before they succeed in their attacks. Most web server attacks start with a malicious user or automated tool probing the website for information leakage or potential vulnerabilities. Subsequently, any attacker attempt to exploit a vulnerability activates an alarm that intercepts the attacker's communications and transmits back misleading information. For example an attempt by the attacker to use boundary input cases on forms and provoke an unintended response from the server causes an embedded code fragment (a code landmine) to fire and activate an alarm.

Deception in the web server relies on several components:

- The application can contain fake code, fake form fields and fake files.
- Landmine code which generates alarms when any attempt is made to access it.
- Spoofing network data sent to attackers
- Dynamic rerouting of attacker traffic for asset protection
- Identifying the attacker's actions based on his interaction with landmine code.
- Recording the attacker's actions after an alarm is activated.
- Interaction with other deployed security technologies.
- The ability to tag attacker's browser so he can be identified in future attacks.

Deceptive activities must attempt to control the attack by sending the attacker fake information such as fake file containing incorrect data, worthless password files or incorrect application responses. Furthermore, deception can be enhanced by increasing the investment in time and effort the attacker need to make to continue the attack. For example, a typical account lockout policy that blocks an attacker after five failed login attempts can be replaced by a deceptive one whereby the attacker is presented with a higher-level authentication after three failed attempts. If he attempts to defeat the authentication, the server assumes that it is under attack and slows down the response to the login attempts. Finally the server allows the attacker to login even with an incorrect password and redirects all traffic to a honeypot. As a further example, an attacker may attempt to manipulate form fields that are protected by landmine code. When unusual behavior generates an alarm, all attacker traffic is redirected to a virtual sandbox created dynamically for the attacker and presenting a fake but believable web site. Similarly, landmine code can cause an alarm when an attacker is attempting to exploit an SQL injection vulnerability. At that point, deceptive code can start leaking other fake database-related data, possibly encrypted but appearing enticing to the attacker, such as password hashes which when broken reveal passwords that either do not work or are associated with fake accounts that feed more fake information.

Decoys can be put to effective use in the protection of a web server through deception [7,8]. Decoys exist as a large collection of fake virtual machines, and detection points: fake server files, fake parameters, fake functions, fake inputs and fake configuration files that appear to end users and attackers as part of the application itself. If an attacker attempts to communicate with a decoy virtual machine the suspicious activity causes an alarm. Similarly, if while probing a web site an attacker touches a detection point an alarm is generated. Detection points include link traversal, such as searching the application for links to hidden resources; attempts to search protected directories; header abuse; illegal request method such as non-standard HTTP methods; input parameter manipulation such as form inputs, injection and cross-site scripting attacks; attempts to manipulate application behavior through query parameter abuse; error codes such as suspicious application errors or unexpected response codes; suspicious file requests such as filenames with known suspicious extensions, prefixes, and tokens; requests for directory configurations, passwords, and protected resources; login attempts with invalid credentials; attempts to crack authentication; and cookie abuse.

For example, in a directory traversal attack, an attacker uses a automated custom spider tool to create a map of all the hidden files and directories that are present on the web server, with the intention of discovering and mining sensitive information such as passwords and configuration settings. These hidden files are not linked from anywhere in the site because they are intended to be accessed by the public, but spider tools can attempt to discover them using a list of common names of these files. A decoy that triggers an alarm when a directory traversal attack is detected can respond that the requested files do

exist and can cause an arbitrary number of fake files and directories to be presented to the attacker, essentially creating a loop for the attacker that can last forever.

Deception in a web server creates a layer of code that forces attackers to reveal themselves but remains invisible to normal users. Once an attacker is identified, he can be tracked or slowed down or blocked. The ultimate effect is supplying the attacker with misinformation, a sequence of fake responses and data to exploit that gives the attacker the impression that he is successful while he is attacking a deceptive server.

Figure 3 shows the architecture of a web server integrated with deception modules. The basic web server functionality draws from the Apache design as described in [9]. 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

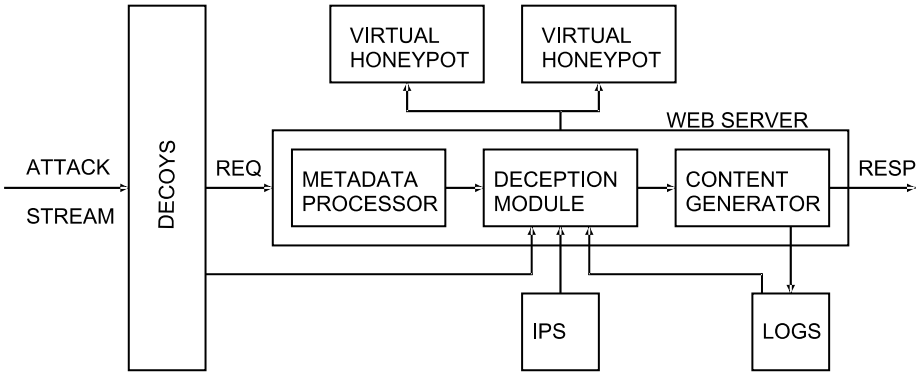

Figure 3. Deceptive web server architecture

referenced by a handler configured by directives in the configuration file httpd.conf. A request goes through several phases in the metadata processor 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. Figure 3 shows a single deception module inserted between the metadata processor and the content generator. However, it is possible to create multiple deception modules and insert them between different Apache phases for even finer control of the deception. Figure 3 shows the deception module communication with the decoys and the IPS to receive alarms and modify the content sent as part of the server's response to the attacker. The deception module may also decide to redirect the attacker traffic to a honeypot when it is determined that the deception has not been successful. For even finer deception control, some of the decoy functionality could become part of the content generator in the form of additional modules.

6 Conclusion

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 architecture of a deceptive web server that integrates intrusion detection, decoys, virtual honeypots and a deceptive content generator that detect the attack, trap the attacker and keep supplying him with misinformation, a sequence of fake responses and data to exploit. Deception technology is thus an additional, increasingly effective, layer of defense against ever more sophisticated attacks that succeed in bypassing the traditional defense layers of firewalls and intrusion prevention systems.

7 References

[1] Bowen, B., et. al., "Botswindler: Tamper resistant injection of believable decoys in vm-based hosts for crimeware detection", Recent Advances in Intrusion Detection, 118-137, 2010.

[2] Cohen, F., Koike, D., "Leading attackers through attack graphs with deceptions", Computers & Security, Volume 22, Issue 5, pp. 402-411, July 2003

[3] Dunnigan , J., & Nofi, A. "Victory and Deceit",

Deception and Trickery in War, 2nd Ed., San Jose, California: Writers Club Press, 2001.
[4] Holz T., Raynal F., "Detecting honeypots and other suspicious environments", Proc. 6th IEEE Information Assurance Workshop, United States Military Academy, West Point, NY, USA, 2005.
[5] http://labrea.sourceforge.net/labrea-info.html
[6] http://www.honeyd.org
[7] http://www.mykonossoftware.com/
[8] http://www.projectnova.org/
[9] Kew, N., "The Apache Modules Book: Application Development with Apache", Prentice Hall, 2007.
[10] Michael J. B., Fragkos G., & Auguston M., "An experiment in software decoy design: intrusion detection and countermeasures via system call instrumentation", Proc. IFIP 18th International [Information Security Conference, Athens, Greece, 2003, 253-264.
[11] Michael, J. B, "On the Response Policy of Software Decoys: Conducting Software-based Deception in the Cyber Battlespace," Proc. 26th Annual International Computer Software and Applications Conference, pp. 957- 962, Aug. 2002.
[12] Michael, J. B., Riehle, R. D., "Intelligent Software Decoys", Proc. Monterey Workshop on Engin. Automation for Software-Intensive Syst. Integration, June 2001, pp. 178-187.
[13] Neagoe, V. , Bishop, M., "Inconsistency in Deception for Defense," Proc. New Security Paradigms Workshop, pp. 31-38, Sep. 2006.
[14] Rowe, N. C., "Deception in defense of computer systems from cyber-attack", in Cyber War and Cyber Terrorism, ed. A. Colarik and L. Janczewski, Hershey, PA: The Idea Group, 2007
[15] Rowe, N. C., Rothstein H., "Two taxonomies of deception for attacks on information systems", Journal of Information Warfare 3 (2) (2004) 27-39.
[16] Voris, J., et. al., "Bait and Snitch: Defending Computer Systems with Decoys", Proc. 3rd Cyber Infrastructure Protection Conference (CIP), Sep. 2012.
[17] https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project