Sebastian Roschke, Feng Cheng, Christoph Meinel: "BALG: Bypassing Application Layer Gateways Using Multi-Staged Encrypted Shellcodes"
in Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011), IEEE Press, Dublin, Ireland, pp. 399-406, 5, 2011. ISBN: 978-1-4244-9219-0.

BALG: Bypassing Application Layer Gateways Using Multi-Staged Encrypted Shellcodes

Sebastian Roschke
Hasso Plattner Institute (HPI)
University of Potsdam
14482, Potsdam, Germany
Email: sebastian.roschke@hpi.uni-potsdam.de

Feng Cheng
Hasso Plattner Institute (HPI)
University of Potsdam
14482, Potsdam, Germany
Email: feng.cheng@hpi.uni-potsdam.de

Christoph Meinel
Hasso Plattner Institute (HPI)
University of Potsdam
14482, Potsdam, Germany
Email: meinel@hpi.uni-potsdam.de

***Abstract*—Modern attacks are using sophisticated and innovative techniques. The utilization of cryptography, self-modified code, and integrated attack frameworks provide more possibilities to circumvent most existing perimeter security approaches, such as firewalls and IDS. Even Application Layer Gateways (ALG) which enforce the most restrictive network access can be exploited by using advanced attack techniques. In this paper, we propose a new attack for circumventing ALGs. By using polymorphic and encrypted shellcode, multiple shellcode stages, protocol compliant and encrypted shell tunneling, and reverse channel discovery techniques, we are able to effectively bypass ALGs. The proposed attack consists of four phases with certain requirements and results. We implemented the initial shellcode as well as the different stages and conducted the practical attack using an existing ALG. The possibility to prevent this attack with existing approaches is discussed and further research in the area of perimeter security and log management is motivated.**

I. Introduction

In the past several years, attacks are becoming more and more sophisticated and intelligent. Many newly emerged attacks are implemented based on the rapidly developed innovative IT techniques, such as advanced cryptography, self-modified code, and integrated attack frameworks, etc., which provides more possibilities to circumvent most available perimeter security approaches. Cryptovirology [9] offers a convenient way to encrypt malicious content or to put backdoors in cryptographic functions. Polymorphic shellcode provides a way for malicious code to modify itself. It is even possible to create shellcode that appears as English written text [13]. Sophisticated but easy-to-use frameworks and tools are now available, which make it possible even for non-professionals to conduct complex attacks, e.g., *ettercap* for Man-In-The-Middle (MITM) attacks [1], etc. Furthermore, many open source software platforms, e.g., metasploit [6], etc., have been developed and widely used in the community to help people easily write their own exploits, customize the existing exploits, as well as compose complex attacks.

On the other side, several perimeter security approaches are developed and deployed in practice to secure hosts and networks, such as firewalls, Intrusion Detection Systems (IDS), and Application Layer Gateways (ALG). Unfortunately, these approaches are neither effective nor efficient for preventing the above mentioned new attack techniques. Firewalls can be easily penetrated by simple tunneling. IDS needs to handle efficient evasion techniques. ALGs provide more restrictions for network access by combining filtering on the application layer and IDS techniques, such as deep packet inspection. Most of ALG implementations provide filtering due to application layer protocol compliance and even allow to block certain commands within a specific protocol. Although ALGs enforce a very restrictive access policy, it is still possible to circumvent such devices by using modern attack techniques.

In this paper, we propose a sophisticated attack for circumventing the security measures introduced by ALGs by using new attacking techniques, such as polymorphic and encrypted shellcode, shellcode stages, protocol compliant and encrypted shell tunneling, and reverse channel discovery techniques. The proposed attack consists of: *Attack Preparation*, *Attack Execution*, *Shellcode Execution*, and *Post-attack Cleanup*. As the major phase of the entire attack process, the *Shellcode Execution* performs complex tasks: *Stage Loading*, *Channel Discovery*, *Connect Back* through Discovered Channel, and *Reverse Shell Tunneling*. Requirements and results of each phase are analyzed and described. We implemented the initial shellcode as well as the different stages and conducted the practical attack using an existing ALG. The possibility to prevent this attack with existing approaches is discussed and some further research works, e.g., log management, are proposed.

The rest of the paper is organized as follows. A short overview about the Application Layer Gateway (ALG) is given in Section II. In Section III, we describe the general concept of attacking ALGs. Section IV describes the actual implementation of the attack using generated shellcodes and existing tools. In Section V, a practical attack scenario is described based on the conceptual idea. After the attack and its implications is discussed in Section VI, the paper is concluded in Section VII.

II. Overview of Application Layer Gateway

Application Layer Gateway (ALG), also simply called application layer proxy, is similar to circuit level gateway except that it is application aware. It can filter packets at the application layer of TCP/IP, as shown in Figure 1. This means it is able to analyze the application data itself and can identify

Sebastian Roschke, Feng Cheng, Christoph Meinel: "BALG: Bypassing Application Layer Gateways Using Multi-Staged Encrypted Shellcodes"
in Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011), IEEE Press, Dublin, Ireland, pp. 399-406, 5, 2011. ISBN: 978-1-4244-9219-0.

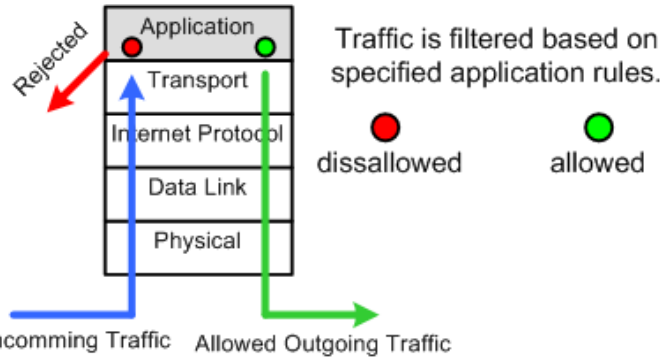

Figure 1. Principle: Application Layer Gateway

malformed application layer data or suspect application commands. Furthermore, ALG can identify if a certain packet is part of the application protocol that should be used through this proxy. ALGs prevent the connection-based tunneling of non desired application protocols, because the proxy will identify that this is not the intended protocol that is allowed to pass through. In plain terms, an ALG that is configured to be a web proxy will not allow any ftp, gopher, telnet or other traffic through, unless it is disguised as such traffic. Because they examine packets at application layer, they can filter application specific commands. This can not be accomplished with either packet filtering or circuit level firewalls, which both do not know anything about the application layer information. Service specific packets, incoming or outgoing, can not pass unless there is a related proxy. ALGs can also be used to log user activities, such as logins. They offer a high level of security, but have a significant impact on connectivity and performance. This is because of context switches that slow down network access dramatically. They are not transparent to end users and require additional configuration of each client computer. Some known ALG implementation includes Microsoft Forefront United Access Gateway (UAG) [21], Siemens Lock-Keeper [16], or Balabit Zorp GPL [19], etc.

Since some malicious data can only be recognized when they are in the form of application layer data, e.g., web page script, file, E-Mail, etc., the *Application Layer Gateway* technology, which is capable of understanding the protocol used by the specific applications that it supports, significantly improves the security of the protected targets by preventing the attacks hiding in the application layer. Furthermore, *Anti-Virus Software*, which usually works on the application layer, is feasible and in practice frequently integrated as an important component of the ALG.

Many known attacks towards the ALG solutions take advantage of the vulnerabilities of the ALG software itself, either the technical failures during the design, implementation and deployment or the human factors due to abuse, misconfiguration, and bad policy. Most of successful application layer attacks penetrate the ALG by exploiting these vulnerabilities and then disabling the security functions. The normal services offered by a compromised ALG will be therefore denied. In this paper, we propose a new attack which targets at the basic principle of the ALG. It can be assumed that the ALG, which we will discuss in the following, is perfectly designed, implemented and deployed.

III. A Generic Attack for Bypassing ALGs

A practical attack ofen has a set of preconditions, e.g., knowledge on the target environment, an existing vulnerability, and a way to store shellcode stages. Although we realized the attack mostly relies on specific vulnerabilities in an existing environment, the attack can be generalized to a certain extend, as it is possible to use different vulnerable programs for the initial exploit and various communication channels that may exist. Besides the general attack we describe in this paper, there are multiple techniques to improve the attack by hiding the established communication channels or deceiving security measurements, such as Intrusion Detection Systems (IDS). Figure 2 shows a basic standard scenario for such kinds of attack. The attacker is located in the external network, which is separated from the internal network by an *Application Layer Gateway* (ALG). The users in the internal and external network are working with specific software and have access to servers providing specific application, e.g., file transfer or mail transfer. The ALG supports only a limited set of application layer protocols and blocks all protocols it is not aware of. In this case, all communication between the internal and external network needs to follow certain rules and to be standard compliant, i.e., generic TCP/UDP communication is not supported.

Similarly, to successfully attack the ALG, there should be several preconditions fulfilled. The first precondition is knowledge on the environment that should be attacked. Although some necessary information can be guessed or gathered, it is helpful to know about the infrastructure that is attacked, e.g. existing firewalls, gateways, etc. Necessary information includes running software on the infrastructure that is vulnerable and exploitable. This information depends on the attacked environment and can be guessed with a certain accuracy, e.g., many clients in companies use Microsoft Windows and related software. Sometimes there are even only a few solutions available to solve a certain task, e.g., there is a limited

Sebastian Roschke, Feng Cheng, Christoph Meinel: "BALG: Bypassing Application Layer Gateways Using Multi-Staged Encrypted Shellcodes"
in Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011), IEEE Press, Dublin, Ireland, pp. 399-406, 5, 2011. ISBN: 978-1-4244-9219-0.

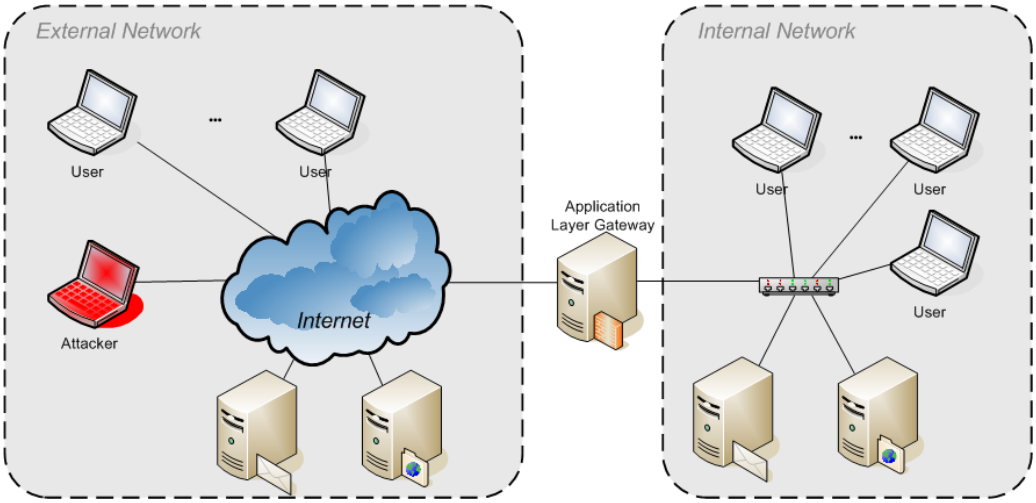

Figure 2. Scenario: Attacking Application Layer Gateway

number of anti-virus solutions for mail gateways available and they can be easily guessed. There needs to be a vulnerable software running within the environment that is supposed to be attacked. Furthermore, this software needs to be exploitable to guarantee a successful attack. These two preconditions might be present in most existing IT environments world wide, which is shown by regular security incidents. The final requirement to perform this attack is a way to store different stages of the shellcode. As this attack requires sophisticated shellcode to be executed, it needs a way to store these shellcodes and execute them afterwards. The sophisticated shellcodes perform complex tasks and are therefore larger than a simple bind shell code block. The shellcode that is injected during the exploit is just a simple loader that calls the different stages sequentially. All the different requirements can be fulfilled in many different ways and are needed for this attack to be successfully executed.

The concrete attack procedure can be shown in the following:

1) Attack Preparation
2) Attack Execution
3) Shellcode Execution
   a) Initial Stage Loading
   b) Channel Discovery
   c) Connect Back through Discovered Channel
   d) Reverse Shell Tunneling
4) Post-Attack / Cleanup

The *Attack Preparation* ensures that all preconditions of the attack are fulfilled. During this phase, a lot of information is gathered on the infrastructure and running software of the target network. It is necessary to find possible attack vectors that can be used to exploit the target network. In this step it is even possible to find a set of attack vectors and to enable the attack to exploit multiple different vulnerabilities which might exist in the target network. Furthermore, the files or data streams to perform the attack are prepared and the stage loading shellcode is stored within. The shellcodes for the different stages are also packed into the attacking files or data streams.

During the *Attack Execution*, the prepared attack files or data streams are sent by the attacker to the target network. This phase provides the attacker with initial control over the system. The stage loading shellcode is executed by modifying the control flow of a target application which is vulnerable to remote code execution, e.g., through a stack or heap based buffer overflow, or a format string handling error in the code. Besides server applications e.g., daemons, the exploitation of client application is likely to be successful, e.g., exploitation of media players or virus scanners through malformed file formats.

The *Shellcode Execution* is used to find and establish a logical communication channel to the internal secured network. It performs many complex tasks, as shown in Figure 3, including: *Stage Loading*, *Channel Discovery*, *Connect Back* through Discovered Channel, and *Reverse Shell Tunneling*. The *Stage Loading* is the initial code execution and is therefore small to fit into an attacked target buffer which might have space restrictions. This shellcode is responsible for loading the other stages and shellcodes from the storage location, e.g., an encrypted file transmitted as a part of the attack. The *Channel Discovery* code is responsible to find a usable channel that can be used to establish a logical connection through the ALG. This can be done by analyzing local network traffic and configurations, and by trying different ways to establish a logical communication channel through the ALG or other gateways connected to the external network. After the discovery of a channel, the *Connect Back* shellcode is executed to establish a logical connection during the *Connect Back* phase. This channel can be established by using file-based communication, mail communication, or any other application layer protocol supported by the ALG. Generic TCP-based connections do not work, as the ALG will prevent the direct connections between

Sebastian Roschke, Feng Cheng, Christoph Meinel: "BALG: Bypassing Application Layer Gateways Using Multi-Staged Encrypted Shellcodes"
in Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011), IEEE Press, Dublin, Ireland, pp. 399-406, 5, 2011. ISBN: 978-1-4244-9219-0.

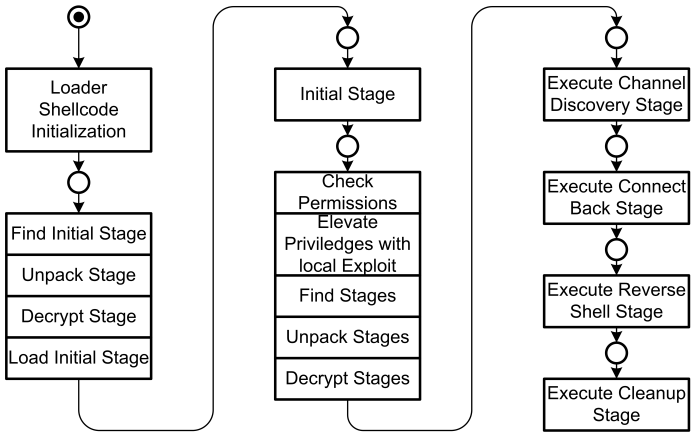

Figure 3. Loader Shellcode and Stages

arbitrary communication partners. When the logical channel is established, the reverse shell can be tunneled using the application layer protocol during the *Reverse Shell Tunneling* phase. In this phase, the amount of transferred messages can be decreased by caching or buffering commands by the shell daemon, sending multiple commands at once, and receiving multiple responses of commands at once. The communication needs to protocol compliant, i.e., the messages exchanged need to look as original messages of the application layer protocol. Using a mail channel requires the shellcode to create real emails that cover the shell communication as payload.

In the *Attack Cleanup* phase, the attacker tries to remove evidence for the successful attack by restarting crashed software, cleaning logs, etc., which makes the victim not be aware of having been attacked.

The described attack shows a basic concept of attacking ALGs that can be improved by several techniques to deceive security measures, such as IDS. The communication used to establish the reverse shell can be encrypted which makes IDS impossible to detect usual shell commands, such as *id*, *ls*, or others. Another possibility is the usage of polymorphic shellcode that decrypts itself prior execution. This makes it impossible to find often used binary patterns for *fork()*, *dup()*, or *connect()* used in shellcodes to establish reverse shell connections. Besides the usage of application layer payload (e.g., the mail body in the mail protocol), it might be possible to use other application specific fields in the protocol that are transfered through ALG. One example might be the *Reply-To* field of an email which might be used to hold any kind of data as long as it is compliant to a standard format. A shell command could be encoded in Base64 and then concatenated by a standard email suffix, such as "@example.com".

IV. Implementations of Shellcode Stages

The core of the implementation is the loader shellcode to find, unpack, decrypt, and load the *Initial Stage*. To implement this shellcode, we used simple x86 template shellcodes [6] that execute a set of commands, as shown in Figure 4. The functionality for finding, unpacking, decrypting, and loading of the *Initial Stage* is realized using simple commands of the OS. The decryption key is a simple password which is encoded in the shellcode. In this way, the loader is very small in comparison to the functionality it implements, i.e., the loader has a size of 263 Bytes for a Linux based OS. The drawback of this solution is the requirement that specific tools are available on the OS, e.g., *find*, *openssl*, and *gzip* on Linux based OS. Figure 3 shows the execution of the different phases. First, the *Loader* shellcode is executed during the exploitation of a vulnerability. The Loader finds the *Initial Stage* code on the system and unpacks it. After the decryption of the *Initial Stage*, it is loaded and executed. The *Initial Stage* is implemented in C using standard system libraries, e.g., *libc* on a Linux based OS. It finds the different stages for unpacking, decryption, and execution. Multiple calls to *exec()* and *system()* are performed and again system tools are used for searching, unpacking, and decryption. The keys for decryption of the stages are encoded in the binary. Furthermore, the *Initial Stage* checks for available permissions and executes local exploits to elevate permissions if necessary. Currently, we support several local vulnerabilities to be exploited, e.g., CVE-2009-2698, CVE-2009-3002, CVE-2009-3001 [4], and several other Linux Kernel vulnerabilities. The stages, i.e., *Channel Discovery Stage*, *Connect Back Stage*, *Reverse Shell Stage*, and *Cleanup Stage*, are implemented in C using standard libraries and system tools. The final shellcode is shown in Figure 5.

The *Channel Discovery Stage* is responsible for finding available communication channels that can be used to pass the ALG. To achieve this, a rootkit [8] is installed on the system to analyze network traffic and local system calls. In this way, it is possible to recognize credentials that may be useful for using a communication channel. Furthermore, the host is scanned for available tools, e.g., *tftp*, *scp*, *mutt*, that can be used to establish a communication channel. To find available channels, the local tool configuration and history is analyzed which may reveal existing channels. After identifying possible tools and channels, the gathered credentials are used with all available channels to discover a suitable one. The

Sebastian Roschke, Feng Cheng, Christoph Meinel: "BALG: Bypassing Application Layer Gateways Using Multi-Staged Encrypted Shellcodes"
in Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011), IEEE Press, Dublin, Ireland, pp. 399-406, 5, 2011. ISBN: 978-1-4244-9219-0.

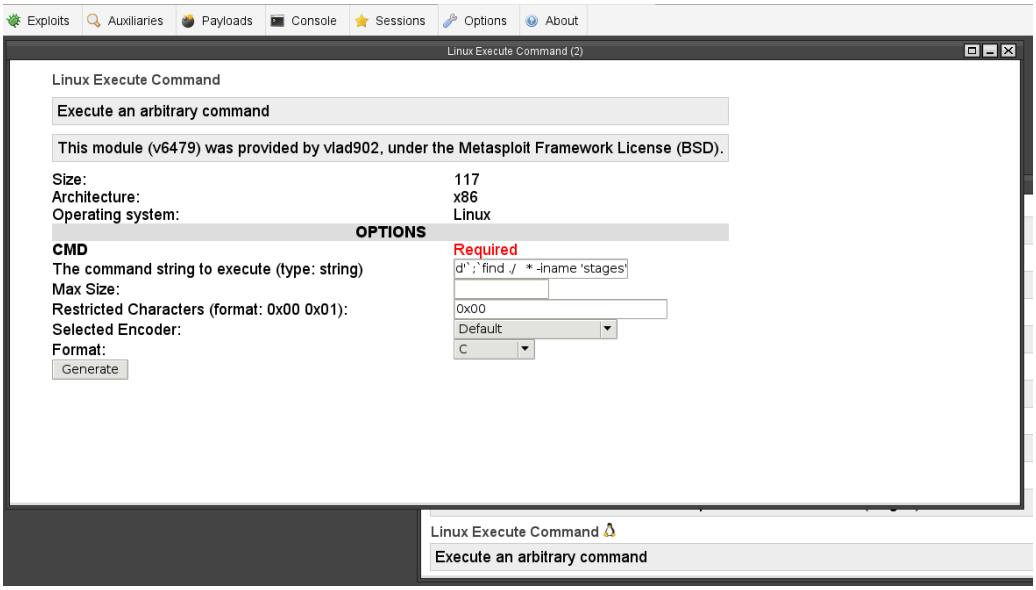

Figure 4. Metasploit Shellcode Creation

most reliable channel we recognized is email, as the usage of this channel does not require many knowledge beforehand, i.e., only credentials, account, and mail server are necessary for using this channel. All this information can be gathered easily with the developed *Channel Discovery Stage*.

After a first channel is established, the *Connect Back Stage* is trying to find the optimal channel for establishing a permanent reverse shell to the target host. The channel should be reliable, should provide a certain throughput, and should yield less evidence that can be used to find this channel. A reliable channel can be established using file transfer via *FTP* [2] or *SCP* [3] protocols. Files provide an easy way to store large information and can be deleted after the information has been used. The permanent reverse shell is established during the *Reverse Shell Stage* using the best available channel. The provides the attacker with direct control over the attacked host using the discovered channel. Furthermore, the attacker can get all information gathered by the installed rootkit. To prevent detection of clear text commands, the communication of the *Reverse Shell Stage* is completely encrypted using the AES block cipher [7]. The *Cleanup Stage* tries to remove all evidence of the attack by cleaning log files and restarting crashed services. This stage can be finalized manually by the attacker. The described stages are using C implemented tools for information gathering, channel discovery, reverse shell establishment. We are using publicly available rootkits for gathering of credentials and analysis of network traffic.

The package for the stages has a size of less than 100 KB. Each stage is encrypted and compressed on its own, and an overall package is created including the *Channel Discovery Stage*, *Connect Back Stage*, *Reverse Shell Stage*, and *Cleanup Stage*. The *Initial Stage* is packaged as the loading shellcode needs to find and extract it easily. The exact size of the shellcode highly depends on the amount of logic that is implemented in the stages. To keep the experiment simple, we just implemented the basic parts that are needed to conduct our experiment successfully. The overall payload size for the concrete implementation used in the experiments is 78.9 KB.

V. A Practical Attack Scenario

As a proof of our concept, we conducted a practical attack on a scenario network secured with an ALG, i.e., *Zorp GPL*. As shown in Figure 6, the internal network consists of several clients and one FTP server. The FTP Server enables public read and write access to specific folders and is configured to scan all its content by an anti-virus software regularly, i.e., *clamav*. The clients have privileged access to the server using *SSH* [3] and can read and write files in directories that are restricted for the public. Unfortunately, the anti-virus software on the FTP server is vulnerable to a buffer overflow described in CVE-2007-6335 [4]. The ALG is configured to only allow traffic that is compliant to the FTP protocol. So the usual shellcode doing a TCP connect back or opening a shell on a specific port can not be used in this scenario. Furthermore, the ALG is configured to detect shell commands in the FTP protocol and block all designated traffic. All clear text shell communication will also be blocked by the ALG. The FTP server is furthermore vulnerable to a local privilege escalation vulnerability described in CVE-2009-2698 [4]. The FTP server is a standard Linux host running Debian Lenny Linux and has multiple common tools installed, e.g., *tftp*, *scp*, *mutt*, *find*, *gzip*, *openssl*, and others. The clients are using Windows XP and have *putty* as *SSH* client installed. The attacker is located in the external network and has access to the public FTP folder.

The attack is conducted by uploading to the public FTP 1) the stage package and 2) a modified Windows executable to trigger the vulnerability. Once the FTP server uses *clamav* to scan the files, the loading shellcode is executed and starting

Sebastian Roschke, Feng Cheng, Christoph Meinel: "BALG: Bypassing Application Layer Gateways Using Multi-Staged Encrypted Shellcodes"
in Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011), IEEE Press, Dublin, Ireland, pp. 399-406, 5, 2011. ISBN: 978-1-4244-9219-0.

```
/*
 * linux/x86/exec - 263 bytes
 * http://www.metasploit.com
 * Encoder: x86/shikata_ga_nai
 * AppendExit=false, PrependChrootBreak=false, CMD=gzip -df
 * `find ./ -iname 'stages.enc.gz'`;openssl enc -d ...;
 * `find ./ -iname 'stages'`, PrependSetresuid=false,
 * PrependSetuid=false, PrependSetreuid=false
 */

unsigned char buf[] =
"\xd9\xee\x2b\xc9\xb1\x3c\xb8\xe7\x91\x6a\x8e\xd9\x74\x24\xf4"
"\x5b\x83\xc3\x04\x31\x43\x13\x03\xa4\x82\x88\x7b\x40\xae\x14"
"\x1d\xc6\xd6\xcc\x30\x85\x9f\xea\x23\x66\xd3\x9c\xb3\x10\x3c"
"\x3f\xdd\x8e\xcb\x5c\x4f\xa6\x05\xa2\x70\x36\xf1\xd9\x19\x46"
"\xdd\x30\xbe\xc0\x3d\x2b\x58\x65\x50\xcf\x84\x5b\x83\x2f\xe8"
"\xca\xb5\x4e\x9f\x69\x6a\xb6\x37\x14\x06\xd4\xa8\xa1\xb9\x56"
"\x5b\x29\x68\xf2\xcd\xd2\x5a\x9b\x6b\x33\xc3\x58\xe3\x4b\x66"
"\xf1\x88\xd8\x04\x2d\x0a\x70\xb6\x0d\xf9\xe8\x18\x63\x60\x95"
"\x2b\x56\x50\x60\xfa\x85\xf7\xe8\x61\xf6\xda\x9f\x04\x9a\x50"
"\x7f\xea\x12\xf8\x0c\x87\xf2\x8a\x93\x14\x80\x50\x62\xe9\x55"
"\x91\xb1\x2d\xb7\xb0\xd7\x0d\xaf\x27\x4b\x22\x40\xdf\xfc\xc8"
"\xf2\x7b\x2c\x48\x65\xe7\x10\xbf\x16\x92\x24\x9f\x80\x39\xa8"
"\xb3\x3f\xb5\x5f\x3e\xac\x5d\xa4\xdd\x44\xf3\xb5\x45\xb5\x20"
"\x31\xa6\xd5\x50\xa8\xc8\x71\xbd\x04\x3a\x5a\x90\x31\x2a\xfb"
"\x87\xa4\x92\xdc\x3f\x43\xbf\x4e\xaf\xfc\x50\xfd\x43\x66\x88"
"\x61\xa7\x06\xb0\x08\xb9\xa2\x1c\xe5\x6a\x0a\x70\x90\x1a\x2b"
"\xe7\x07\xc3\x8c\x9f\xa2\x6f\xbe\x30\x5a\x1f\x4c\xa3\xc0\xf8"
"\xd0\x3b\x5e\x54\x99\xdd\xad\xda";
```

Figure 5. Final Shellcode

to search for the stages. The stages are packaged into a compressed file and encrypted which makes it impossible for the virus scanner to recognize the malicious files. Even if the anti-virus software is executed on the unwrapped stage codes, the only known malicious code in this attack is the rootkit, i.e., the *Mood-Nt* Linux kernel rootkit is slightly modified to fit our purpose. This rootkit exists only in binary format shortly before installation, as it is removed afterwards. The installation of this rootkit is not permanent, which means that it will only reside in kernel memory, so it is difficult to detect by host-based IDS. The installed rootkit will collect all login information of the clients connecting to the compromised FTP server via *SSH*. The channel is established by simple file based communication using the FTP server's public folder. Each command the attacker wants to be executed is wrapped and encrypted in a file, and uploaded to the FTP server. The hidden rootkit is reading files uploaded to the FTP and searches for commands given by the attacker. The commands are executed and the results are wrapped and encrypted again in a file, which the attacker can later download from the FTP server. In this way, the attacker established a logical reverse shell by using FTP compliant functionalities that can not be blocked by the ALG.

VI. Discussion

The basic techniques, such as buffer overflows, format string vulnerabilities, integer overflows, etc., are still used to exploit hosts and are combined with cryptology and new techniques for self-modifying code [5]. In [10], cryptovirology is introduced as the offensive application of cryptography in several attacks. A lot of specific attacks are proposed and described by scientific research, e.g., [12] and [11]. The introduction of polymorphic shellcodes [5] provides further improvement of basic attack techniques by evasion methods to circumvent detection of the malicious code. The recently introduced *English Shellcode* [13] provides a way to hide executable shellcode inside English ASCII text, which makes it difficult to detect, even with statistical measures. Advanced and convenient tools are available to support even unexperienced attackers to conduct attacks, e.g., metasploit [6]. The combination of these techniques and tools to a powerful attack is described in this paper.

To secure networks several perimeter security solutions have been developed, e.g., firewalls, IDS, and ALGs. The basic concept of firewalls [14] provides a way to filter network traffic based on rules describing allowed traffic, by specifying packet parameters, such as destination address, source address, destination port, source port, and others. In addition to the firewall concept, ALGs introduce filtering on application layer and the compliance to certain application layer protocols. Furthermore, it is possible for ALGs to allow only certain commands of a specific application layer protocol (e.g., FTP). Currently, there are several ALG implementations available,

Sebastian Roschke, Feng Cheng, Christoph Meinel: "BALG: Bypassing Application Layer Gateways Using Multi-Staged Encrypted Shellcodes"
in Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011), IEEE Press, Dublin, Ireland, pp. 399-406, 5, 2011. ISBN: 978-1-4244-9219-0.

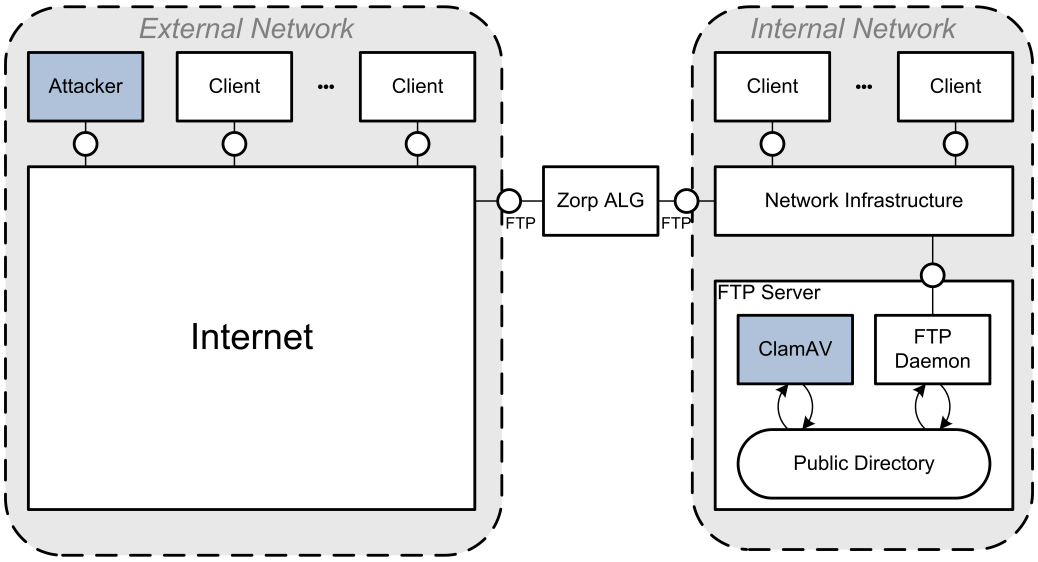

Figure 6. Experiment Scenario

such as Microsoft Forefront United Access Gateway (UAG) [21], Siemens Lock-Keeper [16], or Balabit Zorp GPL [19]. Although these ALGs enforce restrictive access to resources, it is shown that it is possible to circumvent these restrictions with increased effort and advanced shellcodes. Even for IDS sensors which are supposed to be capable of doing deep packet inspection (DPI) [15] it is difficult to detect the attack proposed in the paper, as it uses encoded shellcode for exploitation and encrypted shellcode stages for further attacks. Although the problem of polymorphic shellcode is addressed in (e.g., [23], [24], [22]), encryption provides an almost unpreventable barrier for IDS. Nevertheless, this attack leaves traces in the network that can be analyzed for finding such an attack. Network connections can be logged using netflow data as well as DNS queries. Access to servers (FTP, HTTP, SMTP, etc) can be logged and analyzed. User logins, file access, and program executions can be logged on each workstation in the local network. Although logging can not prevent this kind of attacks in the first place, it provides possibilities for recognizing unusual behavior and network traffic as well as analyzing conducted attacks in detail. Most server and OS implementations provide several logging mechanisms, even mechanisms for remote logging, i.e., storing logs remotely in an automated fashion to circumvent log manipulation by an attacker. Efficient log management is necessary for processing and analyzing huge amounts of logs in realtime. Furthermore, correlation mechanisms can be applied for filtering out security relevant subsets of events. Multiple tools and implementations are available for filtering and analyzing logs, e.g., Sagan as log filtering and correlation tool [20]. IDS correlation techniques and platforms could be directly applied to log analysis. For instance, the steps for IDS correlation described in [18] can be applied to log management in the same way as they are applied to IDS alert management. The platform described in [17] might be used for log correlation and analysis in realtime.

The motivation of this work is to prompt further research on security measures and log management considering advanced attack techniques. The innovation on attack methods and the usage of cryptographic measures for conducting attacks enable the evasion of most recent security measures available. To achieve higher levels of security, it might be necessary to restrict the possibilities for access even further, e.g., by filtering all content that seems to be encrypted or encoded. But as proved with the work on English shellcode, this method also has vulnerabilities as it is undecidable, whether certain data is encrypted, encoded, in plain text, or executable malicious code. Extensive logging might help to detect and analyze these kinds of attacks.

VII. Conclusion

In this paper, we proposed a sophisticated attack method to bypass the security measures introduced by ALGs. Some new techniques, such as polymorphic and encrypted shellcode, shellcode stages, protocol compliant and encrypted shell tunneling, and reverse channel discovery techniques, are applied. The detailed attacking procedure, which includes *Attack Preparation*, *Attack Execution*, *Shellcode Execution*, and *Post-attack Cleanup*, are described. Requirements and results of each attacking phase are analyzed and discussed. An initial shellcode as well as the code for performing the following stages are implemented. A practical scenario is carried out using a real ALG product, the *Zorp GPL* ALG. To our knowledge, it is nearly impossible to prevent the proposed attack by using the existing security approaches. It is expected that further research in the area of perimeter security can be encouraged.

The proposed attack has a set of requirements, such as the existence of standard libraries and tools on the target host, to keep the shellcode as small and efficient as possible. We see

Sebastian Roschke, Feng Cheng, Christoph Meinel: "BALG: Bypassing Application Layer Gateways Using Multi-Staged Encrypted Shellcodes"
in Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011), IEEE Press, Dublin, Ireland, pp. 399-406, 5, 2011. ISBN: 978-1-4244-9219-0.

the possibility to remove these requirements by implementing further functionality in the initial shellcode as well as in the different stages. The stage implementations can also be compiled and linked statically, to make it independent from a specific library. However, this approach would require more existing storage space, as the shellcode and stage implementation will become larger.

REFERENCES

[1] Ettercap: Website: http://ettercap.sf.net/ (accessed Mar 2010).
[2] File Transfer Protocol: FTP Request For Comments - RFC 959, Website: *http://www.rfc-editor.org/rfc/rfc959.txt* (accessed Mar 2010).
[3] Secure Shell Protocol: SSH Request For Comments - RFC 4253, Website: *http://www.ietf.org/rfc/rfc4253.txt* (accessed Mar 2010).
[4] Mitre Corporation: "Common vulnerabilities and exposures", CVE Website: http://cve.mitre.org/ (Accessed March 2009).
[5] J. Koziol, D. Litchfield, D. Aitel, C. Anley, S. Eren, N. Mehta, R. Hassell: "Shellcoders Handbook", Wiley Publishing, Inc. (2004).
[6] The Metasploit Project, Website: http://www.metasploit.org/ (accessed Mar 2010).
[7] NIST AES: "Specification of the Advanced Encryption Standard (AES)", Federal Information Processing Standards Publication 197, November 26, 2001, Website: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf (accessed Mar 2010).
[8] G. Hoglund and J. Butler: "Rootkits: Subverting the Windows Kernel", Addison-Wesley Professional (2005).
[9] A. Young, M. Yung: "Malicious Cryptography: Exposing Cryptovirology", John Wiley & Sons (2004).
[10] A. Young, M. Yung: "Cryptovirology: Extortion-Based Security Threats and Countermeasures", In: Proceedings of the *IEEE Symposium on Security and Privacy (S&P'96)*, Oakland, CA, USA, IEEE Press, pp. 129-141 (1996).
[11] A. Young, M. Yung: "Malicious Cryptography: Kleptographic Aspects", In: Proceedings of the *RSA Conference (RSA'05)*, Springer LNCS, vol. 3376, pp. 7-18 (2005).
[12] A. Young, M. Yung: "Deniable Password Snatching: On the Possibility of Evasive Electronic Espionage", In: Proceedings of the *IEEE Symposium on Security and Privacy (S&P'97)*, Oakland, CA, USA, IEEE Press, pp. 224-235 (1997).
[13] J. Mason, S. Small, F. Monrose, G.G. MacManus: "English shellcode" In: Proceedings of the *16th ACM Conference on Computer and Communications Security (ACM CCS'09)*, Chicago, Illinois, USA, ACM Press, pp. 524-533 (2009).
[14] W. Noonan, I. Dubrawsky: "Firewall Fundamentals", Cisco Press (2006).
[15] S. Northcutt: *Network Intrusion Detection - An Analyst's Handbook*, New Riders (1999).
[16] F. Cheng, Ch. Meinel: "Research on the Lock-Keeper technology: Architectures, applications and advancements", In: *International Journal of Computer & Information Science (IJCIS)*, Plenum Press, New York, NY, USA, vol. 5(3), pp. 236-245 (2004).
[17] S. Roschke, F. Cheng, Ch. Meinel: "A Flexible and Efficient Alert Correlation Platform for Distributed IDS", In: *Proceedings of 4th International Conference on Network and System Security (NSS'10)*, IEEE Press, Melbourne, Australia, pp. 24-31 (September 2010).
[18] R. Sadoddin, A. Ghorbani: *Alert Correlation Survey: Framework and Techniques*, In: Proceedings of the International Conference on Privacy, Security and Trust (PST'06), ACM Press, Markham, Ontario, Canada, pp. 1-10 (2006).
[19] Balabit IT-Security: "Zorp GPL", Website: *http://www.balabit.com/network-security/zorp-gateway/gpl/* (accessed Mar 2010).
[20] Softwink, Inc: "Sagan", Website: *http://sagan.softwink.com/* (accessed Dec 2010).
[21] Mircosoft, Inc.: "Microsoft Forefront United Access Gateway (UAG)", Website: *http://www.microsoft.com/forefront/unified-access-gateway/en/us/default.aspx* (accessed Mar 2010).
[22] M. Talbi1, M. Mejri, A. Bouhoula: "Specification and evaluation of polymorphic shellcode properties using a new temporal logic", In: *Journal in Computer Virology*, Springer, Paris, vol 5(3), pp. 171-186 (2009).
[23] D. Kim, I. Kim, J. Oh, J. Jang: "Tracing Stored Program Counter to Detect Polymorphic Shellcode", In: Journal *IEICE Transactions 2008*, Oxford University Press, vol. 91-D(8), pp. 2192-2195 (2008).
[24] Y. Song, M. E. Locasto, A. Stavrou, A. D. Keromytis, S. J. Stolfo: "On the infeasibility of modeling polymorphic shellcode", In: Proceedings of the *ACM Conference on Computer and Communications Security (ACM CCS'07)*, Alexandria, Virginia, USA, ACM Press, pp. 541-551 (2007).