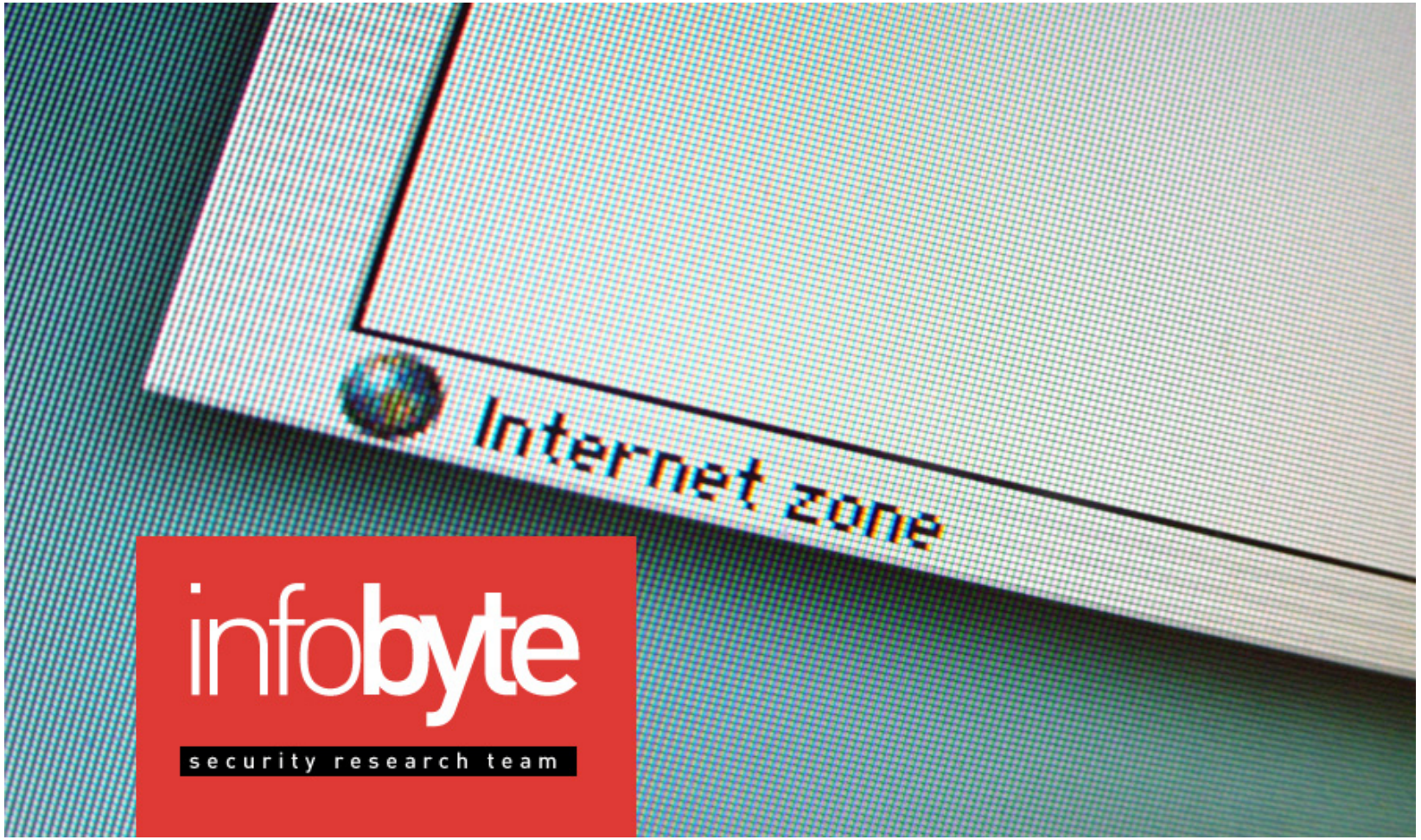


Francisco Amato / Federico Kirschbaum

evilgrade, *"You have pending upgrades..."*

http://www.infobyte.com.ar

Introduction

Topics

- Client side explotation
- Update process
- Poor implementation of update processes
- Attack vectors
- evilgrade framework presentation

Client side explotation

Searching the Weakest Link

Bypassing the fortress walls

This technique allows for example transform a user terminal in a “proxy” to access the internal network of a company

General application's update process

How does it works?

- Update process are either manual or automatic.
- The process requests a special file in the master server for example *update.application.com/info.xml*
- The file has the internal information of the available updates.
- It's installed automatic or ask if you like to install the new update.

What’s the problem?

Is there any problem?

Trust

- A lot of application don’t verify the updates contents.
- They blindly trust without verification of the master update server.

Tool Information

evilgrade is modular framework that allow us to take advantage of poor update implementations by injecting fake updates.

- It’s a opensource project
- It’s developed in Perl

How does it work?

It works with modules, each module implements the structure needed to emulate a false update of specific application.

evilgrade needs the manipulation of the victims's dns traffic

Normal update process

1. App1 start the update process
2. Consult to the dns server host *update.app1.com*
3. DNS server replies *200.1.1.1*
4. App gets the file *lastupdate.xml* from *update.app1.com*
5. App analyzes the update file and detect a new update
6. App1 downloads and execute the update *http://update.app1.com/update.exe*

Attack example

1. App1 starts the update process
2. Consult to the dns server host *update.app1.com*
3. The attacker modifies the DNS traffic and returns other ip address, controlled by the attacker.
4. App1 get the file controlled by the attacker *http://update.app1.com/lastupdate.xml*
5. App1 processes the file and detect a new update
6. App1 downloads and execute the backdoor *http://update.app1.com/backdoor.exe*

Attack vectors?

Possibilities:

Internal scenery:

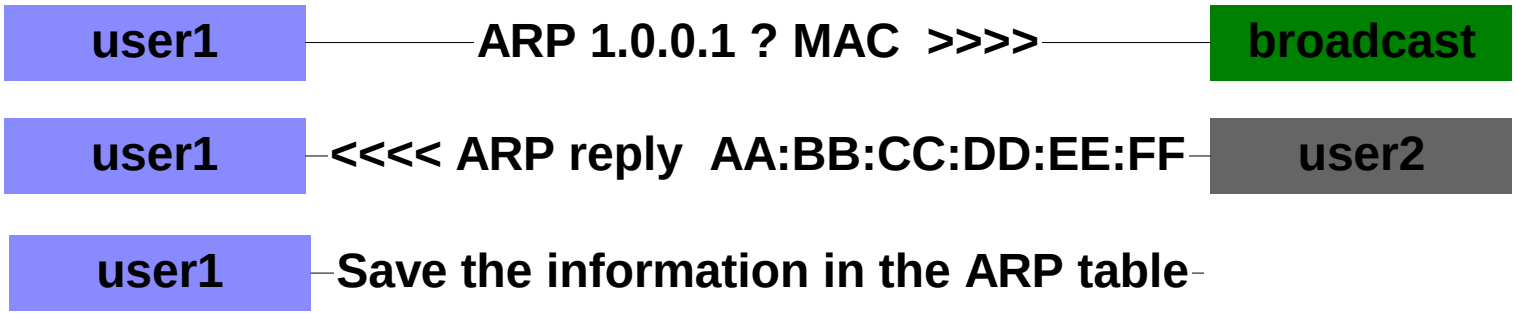
- *Internal DNS access.*
- *ARP spoofing.*
- *DNS Cache Poisoning.*

External scenery:

- *Internal DNS access.*
- *DNS Cache Poisoning.*

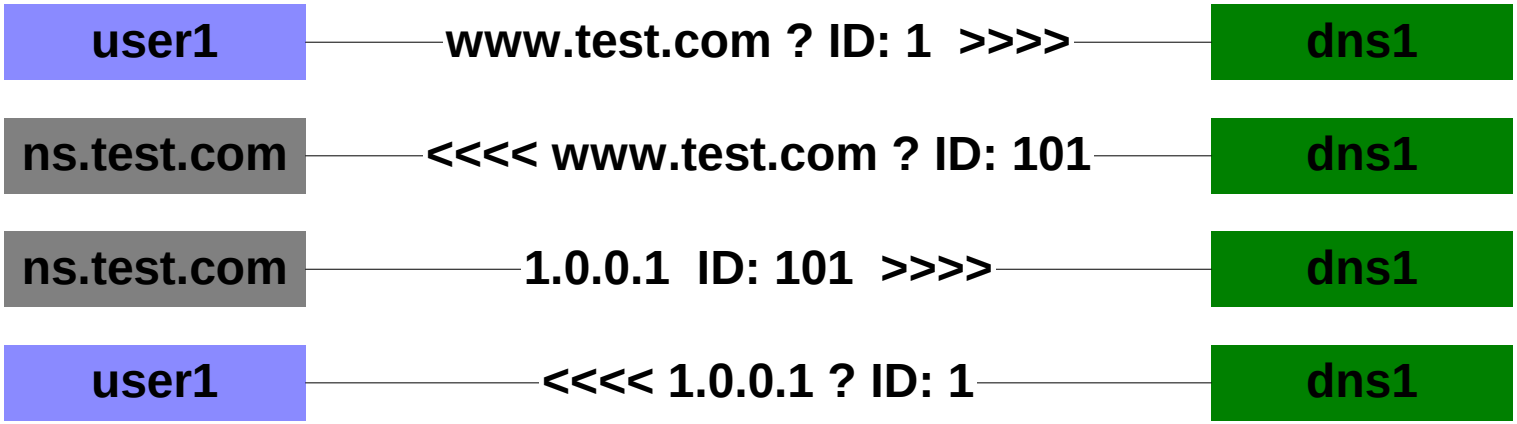
Description

Layer 2 traffic re-routing (MITM)



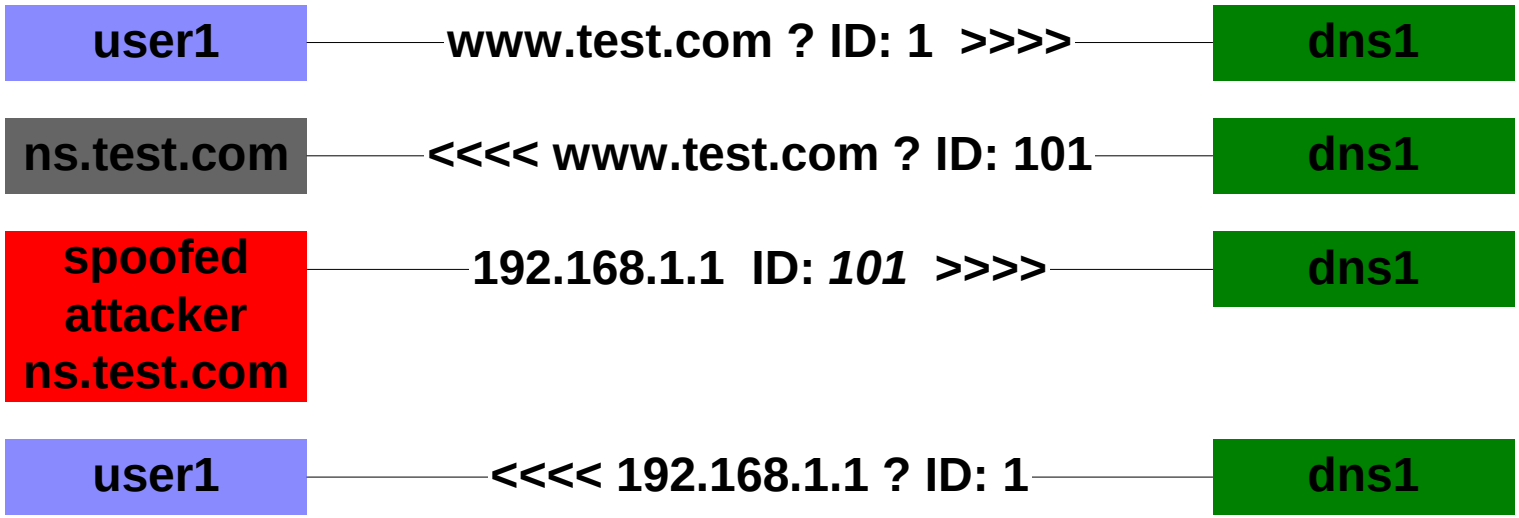
DNS Request

Description



DNS Cache poisoning

Attack



Nothing is easy

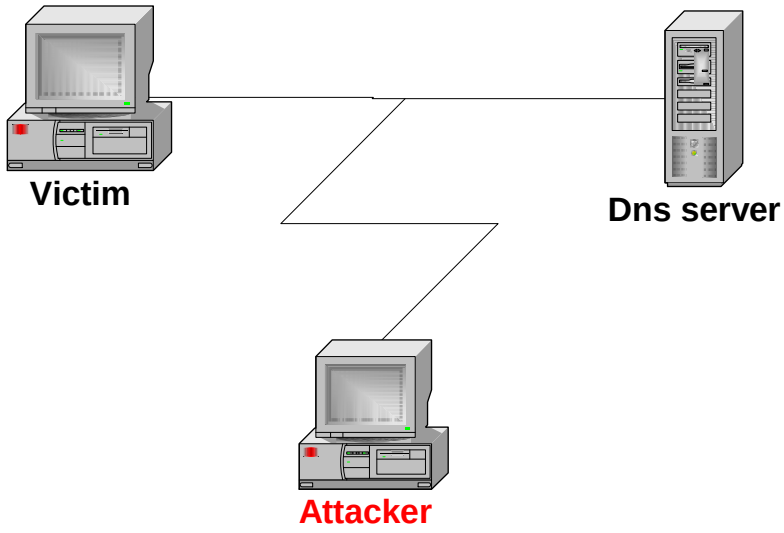
Taking care of:

- TTL.
- Cache.
- Legitimizes response.

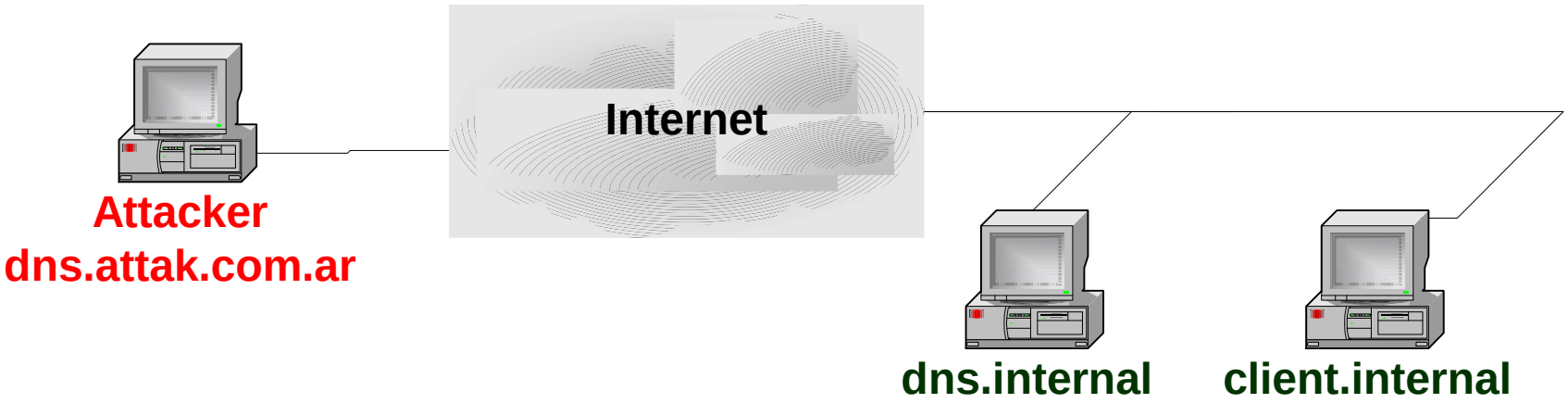
Needed information:

- Source.
- ID 16 bits (65535 possibilities).

Sample Topology



Sample Topology



Is this new?

No, it's not. ☹

The idea of the framework is the **centralization** and **explotation** of different update implementations all together in one tool.

What are the supported OS?

The framework is multiplatform, it only depends of having the rigth payload for the platform to exploit.

What can I do with it?

This attack vector allows the injection of fake updates to remotely access a target system.

Console:

It works similar to a IOS console:

-**show <object>:** Used to show different information.

-conf <object>: Enter to the configure mode.

-set <option> “value”: Configures different options.

-start: Webserver starts.

-stop: Webserver stops.

-status: Webserver status.

Modules:

```
package modules::sunjava;

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

my $base=
{
    'name' => 'Sun Microsystems Java',
    'version' => '1.0',
    'appver' => '< 1.6.0_03',
    'author' => [ 'Francisco Amato < famato +[AT]+ infobyte.com.ar>' ],
    'description' => qq{},
    'vh' => 'java.sun.com',
    'request' => [
                    {
                    'req' => '^/update/[.\d]+/map\-[.\d]+.xml', #regex friendly
                    'type' => 'file', #file|string|agent|install
                    'method' => '', #any
                    'bin'    => '',
                    'string' => '',
                    'parse' => '',
                    'file' => './include/sunjava_map.xml'
                    },
                    {
                    'req' => '^/java_update.xml$', #regex friendly
                    'type' => 'file', #file|string|agent|install
```

Request:

It’s an object’s collection.

Each object it’s a possible HTTP request inside the virtualhost configured for the module.

Request:

Each object has:

<**req**> - requeried URL (regex friendly).

<**type**> : [file | string | agent | install]

<**method**> : [GET|POST|TEST|””]

<**bin**> : [1|””] If is it a binary file.

<**string**> : String request’s response

<**parse**> : [1|””] If this file or string need be parsed

<**file**> : The path of the request’s response

Options:

```
'options' => {  'agent'   => { 'val' => './agent/reverseappsign.exe',
                               'desc' => 'Agent to inject'},
                'arg'     => { 'val' => '',
                               'desc' => 'Arg passed to Agent'},
                'enable' => { 'val' => 1,
                               'desc' => 'Status'},
                'name'   => { 'val' =>
                          "'javaupdate'.isrcore::utils::RndAlpha(isrcore::utils::RndNum(1))",
                            'hidden' => 1,
                            'dynamic' =>1,},

                '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'},
                'atitle'   => { 'val' => 'Critical vulnerability',
                                'desc' => 'Title name display in the systray item pop'},
                'adescription' => { 'val' => 'This critical update fix internal vulnerability

                                    'desc' => 'Description display in the systray item pop'},
                'website' => { 'val' => 'http://java.com/moreinfolink',
                               'desc' => 'Website display in the update'}
            }
```

Agent:

Agent is the fake update to be injected in the victims's computer.

Implemented modules:

- Java plugin
- Winzip
- Winamp
- MacOS
- OpenOffices
- iTunes
- linkedin toolbar
- DAP (download accelerator)
- notepad++
- speedbit



Time for the demo.
Cool!

A more secure approach

- Update server running under https, certificate control.

- Digital signatures, verify the update with a public key

and you know..

Next time you do an update!

don’t believe in everything you see

infobyte

More Info

- http://www.secureworks.com/research/articles/dns-cache-poisoning/#update
- http://www.trusteer.com/docs/bind9dns.html
- http://www.trusteer.com/docs/bind8dns.html
- http://en.wikipedia.org/wiki/ARP_spoofing
- http://www.trusteer.com/docs/microsoftdns.html

Questions!

???

Thanks!

Contact

blog.infobyte.com.ar

Francisco Amato – famato@infobyte.com.ar

Federico Kirschbaum – fedek@infobyte.com.ar