The Metabrik Platform - Rapid Development of Reusable Security Tools

Patrice Auffret

metabrik.org

October 2016

Whoami?

GomoR

- Information security engineer for 15 years
- *Perl* developper for same amount of time
- CPAN author
  - *Net::Packet* (obsolete)
  - *Net::Frame* suite (successor)
  - *Net::Write*
  - *Net::SinFP/Net::SinFP3* (hack.lu, ekopary, EuSecWest 2012)
  - *Metabrik* (and *Metabrik::Repository*)
- Interests
  - Forensic analysis
  - Network protocols
  - Big data

What is Metabrik?

The subject of today...

A Pokemon?

Yes, but no.

- Source: http://www.mypokecard.com/fr/Galerie/Pokemon-Metabrik

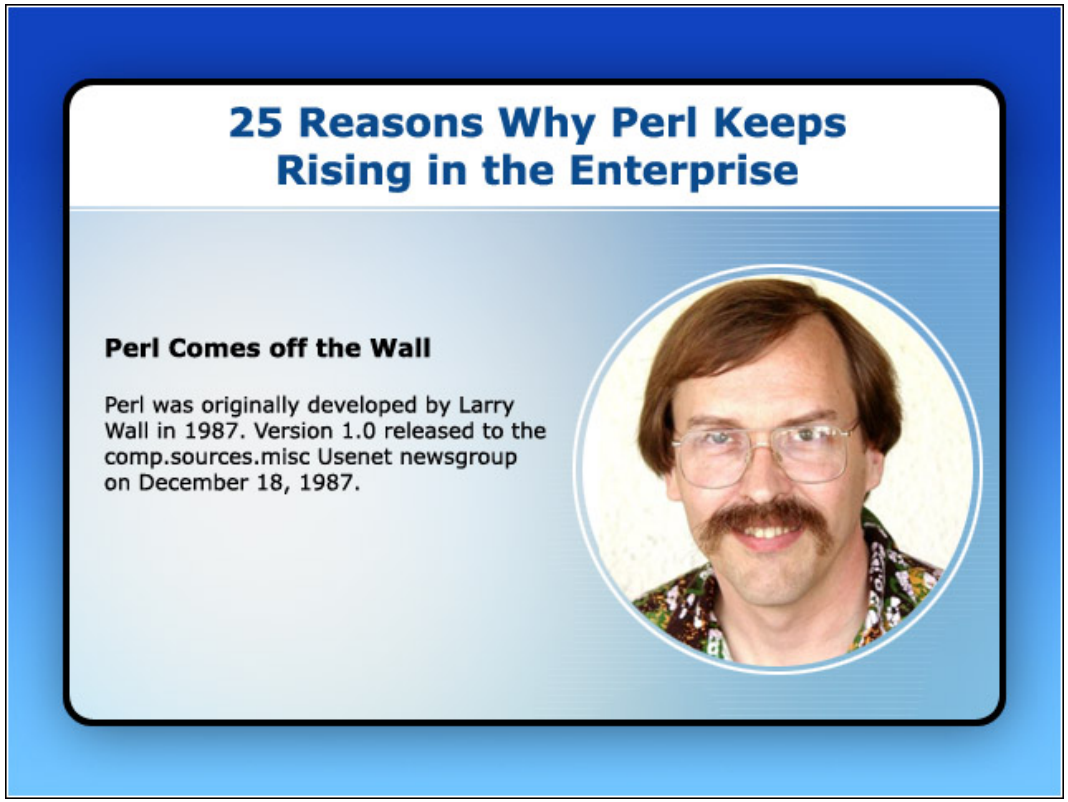
A platform

And much more.

- A *UNIX*-like shell
- A *true* language with a *Read-Eval-Print-Loop*
- Many *Brik*s
  - A development/prototyping platform
  - To build quickly the right tool

A *true* language?

Yes.



- Source: eweek, 30 Avril 2010

Why?

CLI rules

- Everything should be possible via command line
  - Automate all the things
- *Do it once* principle
  - Tired of repeated throw-away scripts
  - Code reusability rules
- *UNIX* shells too simple
  - *Pipe* too limited
  - Needed a powerful language
- Rapid development from a *CLI*
  - Writing scripts is also possible
- Normalized Syntax in *human readable* form

Comparison with REbus

Interactive usage versus fully automatic one

- Same goal: normalize tool usage
- REbus: replace the human
  - Not a shell
  - Usage is not so easy
  - Written in *Python*
  - Input/output automation
- Metabrik: help the human
  - Manual input/output
  - Easy *Brik* usage
- Both are using *wrapper*s around existing tools
  - Metabrik does not only use *wrapper*s
  - It tries to use best *Perl* CPAN modules

Demo 1 - The Metabrik Shell (~3 minutes)

- 3 kinds of command lines
  - external
  - *Brik*'s commands
  - *Perl* code (*REPL*)
- 5 *Brik*'s commands
  - *use, set, get, run, help*

Features

Most notable ones

- Shell: builtins
  - cd, alias, ...
- Customizable shell with history handling
  - File *.metabrik_rc*
  - File *.metabrik_history*
- *Metasploit*-like syntax
- Completion for commands, files, variables
- Control keys like *Ctrl+R*

Briks

You have the knowledge, detail is in the *Brik*

- Many *wrapper*s around external programs
  - but not only
- Object-oriented
  - defined by properties like tags, category or attributes
  - a *Brik* may inherit from one or more others
  - Example: *file::psv* inherits from *file::csv*
- Add features to existing tools
  - There is always one lacking
  - Example: *forensic::scalpel*
- A reusable *Perl* module
  - a command line interface
  - a classic interface

brik::tool Brik

Makes a *Brik* easy to use

- Two kind of dependencies
  - System packages
  - *Perl* modules
- Easy to install and use
  - *run brik::tool install database::cvesearch*
  - *use database::cvesearch*
  - *help database::cvesearch*
- Easy to update as
  - *run brik::tool update*

Special variables

Where we keep some *Perl* philosophy

- Example:
  - *run shell::command capture ls /*
  - *my $count = scalar(@$RUN)*
- Input/output handling via $RUN
  - This is the new pipe
  - *Perl* basic data types
  - You sculpt it to your needs
- Other special variables
  - *$SET*
  - *$GET*
  - *$CON, $LOG, $GLO, $SHE*
  - *$USE, $ERR, $MSG, $REF*

Demo 2 - forensic challenge (~3 minutes)

Or how to quickly solve a problem

- Some miscreants kidnapped your cat
- We found an old device on crime-scene
- We have to analyze this data
- File analysis
  - *file::type*
  - *file::compress*
  - *image::exif*
- Extract data
  - *forensic::scalpel*

Metabrik Core and Metabrik Repository

What's the difference?

- Metabrik Core
  - *core::global*
  - *core::shell*
  - *core::log*
  - *core::context*
  - Minimal system and *Perl* modules dependencies
- Metabrik Repository
  - 200+ *Brik*s (and counting)
  - *brik::tool* to manage
  - *brik::search* to search
  - Install dependencies only when needed

brik::tool and brik::search

Your best friends

- Search by tag, command, category or string...
  - *run brik::search tag video*
- *brik::tool* for management
  - create a skeleton of a new *Brik*
    - *run brik::tool create_brik my::first*
  - create a skeleton of a new program
    - *run brik::tool create_tool my_tool.pl*

A *Metatool*

From prototype to industrialisation

- Finalized prototype
  - *run brik::tool create_tool iplocation.pl*
- Shell commands conversion to code

```
1  # Shell Metabrik
2  use lookup::iplocation
3  run lookup::iplocation from_ip 93.184.216.34
```

```
1  # Perl program
2  use Metabrik::Core::Context;
3  my $con = Metabrik::Core::Context->new;
4
5  use Metabrik::Lookup::Iplocation;
6  my $li = Metabrik::Lookup::Iplocation->new_from_brik_init($con);
7  my $h = $li->from_ip($ip);
```

Demo 3 - automate malware analysis (~3 minutes)

Or how to extract *Indicators of Compromise*

- Use a *VM* as a scapegoat (sacrifice it)
- Take a fingerprint of its memory/process/registry before
- Run a malware
- Take a fingerprint of its memory/process/registry after
- Instrumentalise a *VM* and take a snapshot
  - *system::virtualbox*
- Execute program remotely
  - *remote::winexe*
  - *remote::wmi*
- Perform a diff on a *Windows* machine-state
  - *forensic::volatility*

Demo 4 - bind all *Brik*s together (~5 minutes)

Make it straightforward

- Use all previous *Brik*s to write a new one
- *remote::windiff*
- Automates diffing between two *VM* states

Enlarge your tools

Use more *Briks*

- Code: *lib/Metabrik/Remote/Windiff.pm*
- Improve a tool by yourself from the 200+ *Briks*
  - *run file::csv write $process_diff out.csv*
  - *run client::dns ptr_lookup $ip*
  - *run api::virustotal ipv4_address_report $ip*
  - *run api::shodan host_ip $ip*

Some of the best *Briks*

For some catagories

- api::*
  - splunk, ...
- client::*
  - elasticsearch, mongodb, redis, rest, openssh, twitter, splunk, ...
- server::*
  - rest, snmp, dns, ...
- proxy::*
  - http, ssh2tcp, ...
- www::*
  - shorten, google, ...
- lookup::*
  - iplocation, oui, ...
- network::*
  - nmap, linux::iptables, sinfp3, ...

Conclusion

- More than 200 *Brik*s...
- Everything becomes a *Perl* variable
- Automate all the things from *CLI*
- Add missing features to existing tools
- Normalization brings reusability and consistency
- Shell unification
- POLL: who would be interested in a workshop?

Question(s)?

Metabrik
There is a Brik for that.

- Code available on: *http://trac.metabrik.org/*
- Howto install: *https://www.metabrik.org/metabrik/install/*
- Docker: *docker pull metabrik/metabrik* (this is for @xme)
- Twitter: @Metabrik
- Twitter: @PatriceAuffret