Introduction to Parasitic Computing with

BrainSlug

Parasitic Computing

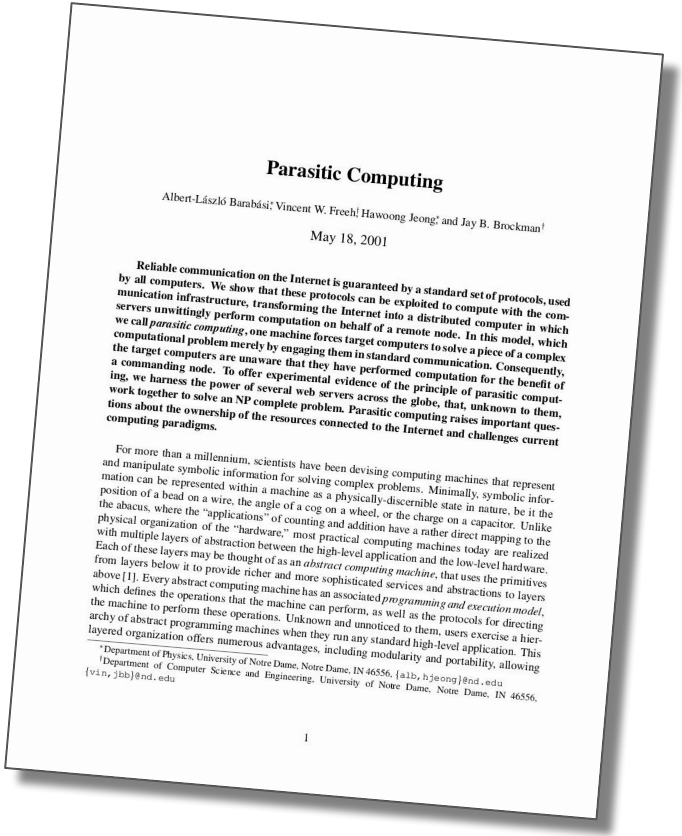
Definition

A **programming technique** where a program in normal authorized interactions with another program **manages to get the other program to perform computations** of a complex nature.

First proposed in 2001...

...by Department of Physics and Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, Indiana

They managed to solve a large and extremely difficult **3-SAT** parasitizing the **TCP** stack of remote web server



Parasitic Computing

Albert-László Barabási[*], Vincent W. Freeh[†], Hawoong Jeong[*], and Jay B. Brockman[†]

May 18, 2001

Reliable communication on the Internet is guaranteed by a standard set of protocols, used by all computers. We show that these protocols can be exploited to compute with the communication infrastructure, transforming the Internet into a distributed computer in which servers unwittingly perform computation on behalf of a remote node. In this model, which we call *parasitic computing*, one machine forces target computers to solve a piece of a complex computational problem merely by engaging them in standard communication. Consequently, the target computers are unaware that they have performed computation for the benefit of a commanding node. To offer experimental evidence of the principle of parasitic computing, we harness the power of several web servers across the globe, that, unknown to them, work together to solve an NP complete problem. Parasitic computing raises important questions about the ownership of the resources connected to the Internet and challenges current computing paradigms.

For more than a millennium, scientists have been devising computing machines that represent and manipulate symbolic information for solving complex problems. Minimally, symbolic information can be represented within a machine as a physically-discernible state in nature, be it the position of a bead on a wire, the angle of a cog on a wheel, or the charge on a capacitor. Unlike the abacus, where the "applications" of counting and addition have a rather direct mapping to the physical organization of the "hardware," most practical computing machines today are realized with multiple layers of abstraction between the high-level application and the low-level hardware. Each of these layers may be thought of as an *abstract computing machine*, that uses the primitives from layers below it to provide richer and more sophisticated services and abstractions to layers above [1]. Every abstract computing machine has an associated *programming and execution model*, which defines the operations that the machine can perform, as well as the protocols for directing the machine to perform these operations. Unknown and unnoticed to them, users exercise a hierarchy of abstract programming machines when they run any standard high-level application. This layered organization offers numerous advantages, including modularity and portability, allowing

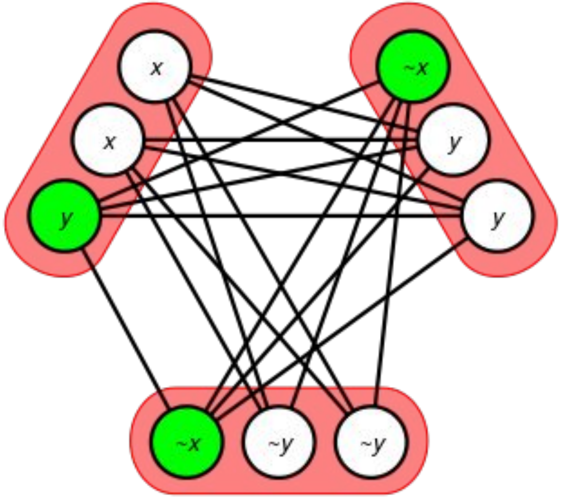
[*]Department of Physics, University of Notre Dame, Notre Dame, IN 46556, {alb,hjeong}@nd.edu

[†]Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, {vin,jbb}@nd.edu

1

What is a 3-SAT Problem?

$$(l_1 \lor l_2 \lor x_2) \land (\neg x_2 \lor l_3 \lor x_3) \land (\neg x_3 \lor l_4 \lor x_4) \land \cdots \land (\neg x_{n-3} \lor l_{n-2} \lor x_{n-2}) \land (\neg x_{n-2} \lor l_{n-1} \lor l_n)$$



Given a **complex boolean expression**, formed by variables, *AND*, *OR*, *NOT* and parentheses.

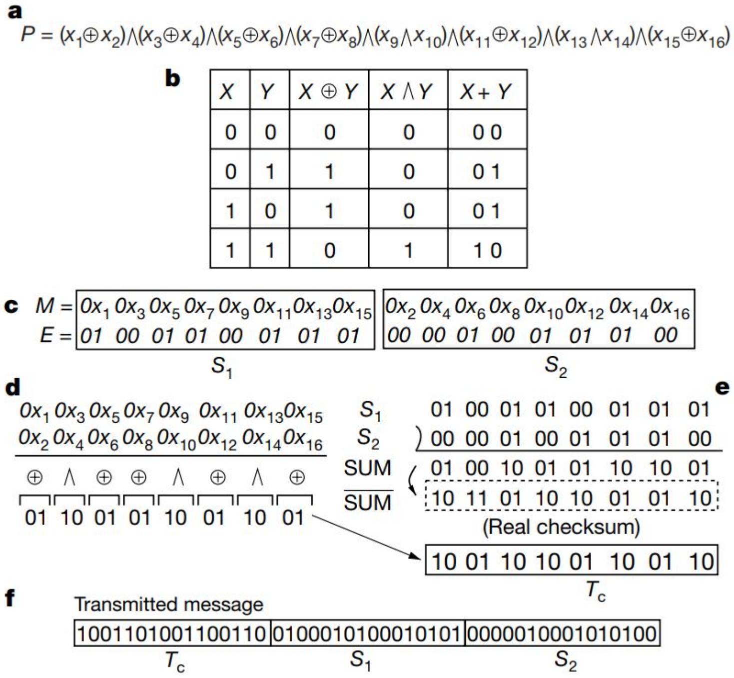
Find **if there is any combination of values** for those variables to make the whole expression evaluate to *TRUE*.

The more difficult 3-SAT problem is known to be NP complete.

...one property of the TCP checksum function is that it forms a sufficient logical basis for implementing ***any Boolean logic function****, and by extension, any arithmetic operation...*

Computing with TCP Checksum

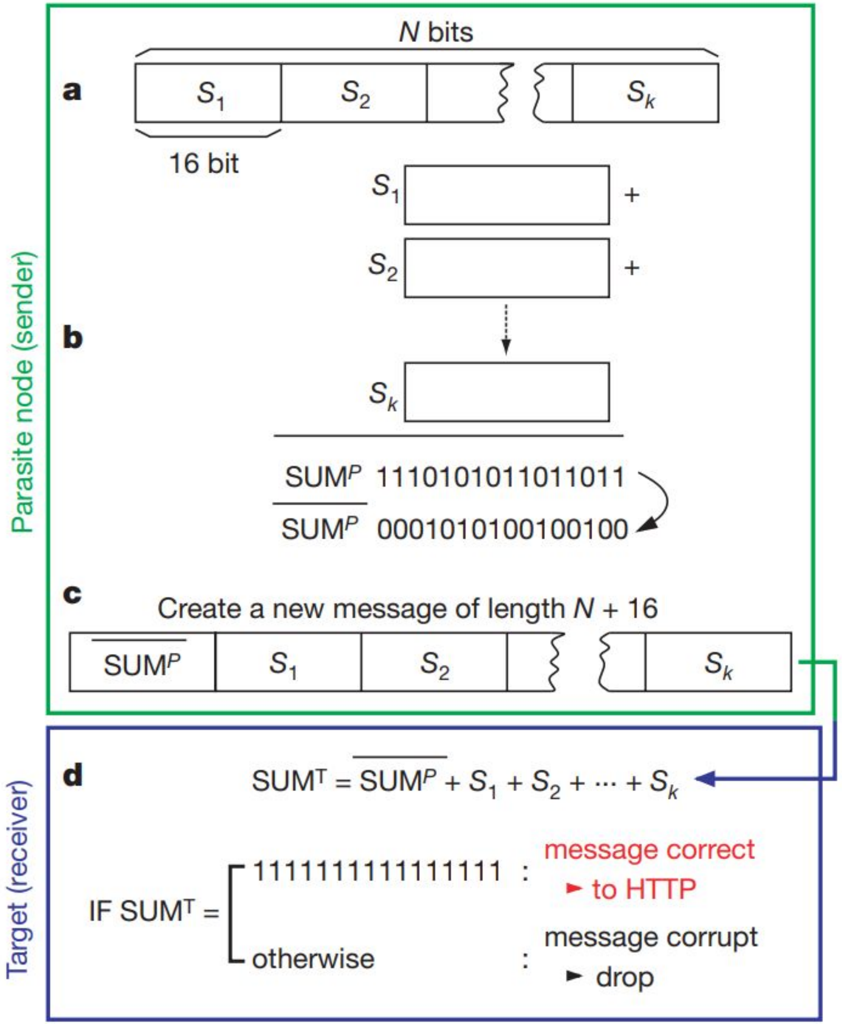
The original 3-SAT problem is decomposed in a considerable number of smaller problems that can be solved with the capabilities of the TCP Checksum function



TCP Checksum Encoding

A TCP packet can be forged to include a guess for the subset of the problem to be checked by the checksum function.

If the guess is correct the packet would be valid and vice versa.



Step 1: Open "modified" TCP connection

1a) Send SYN message

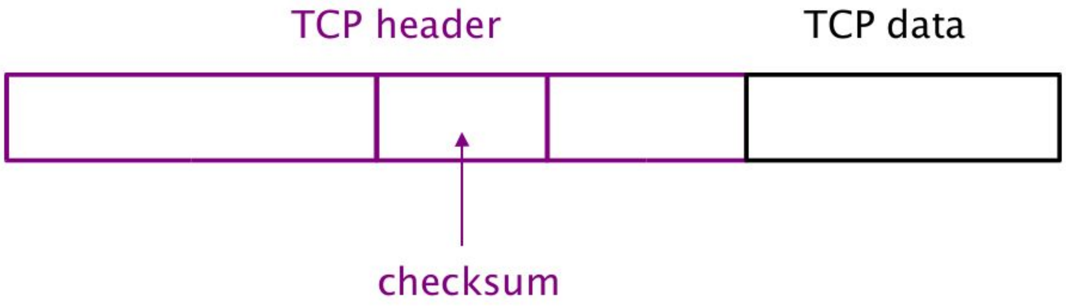
Step 1: Open "modified" TCP connection

1b) Receive SYN message,
extract host sequence number

Step 1: Open "modified" TCP connection

1c) Send ACK message,
connection open, ready for data

Step 2: Prepare TCP segment



- Checksum
  - Determined by SAT equation
- Data
  - Values of variables for this test

Step 2: Prepare TCP segment

- Compute checksum
  - Normally
    - Put 0's in checksum field
    - Sum segment (add each 16-bits)
    - Insert complemented sum into checksum field
  - Modified (for our exploit)
    - Put 0's in checksum field
    - Put "answer" in data field, padded to proper length
    - Sum segment (add each 16-bits)
    - Insert complemented sum into checksum field

Step 3a: Compute (positive answer)

3.a.1) Send modified TCP segment

Step 3a: Compute (positive answer)

3.a.2) TCP segment is valid,
pushed up to HTTP

Step 3a: Compute (positive answer)

3.a.3) Receive HTTP reply
interpret this as a positive answer

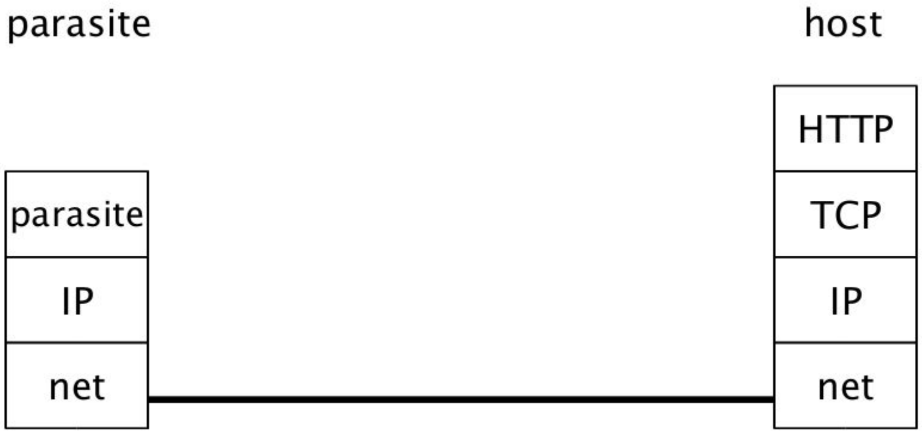
Step 3b: Compute (negative answer)

3.b.1) Send modified TCP sement

Step 3b: Compute (negative answer)

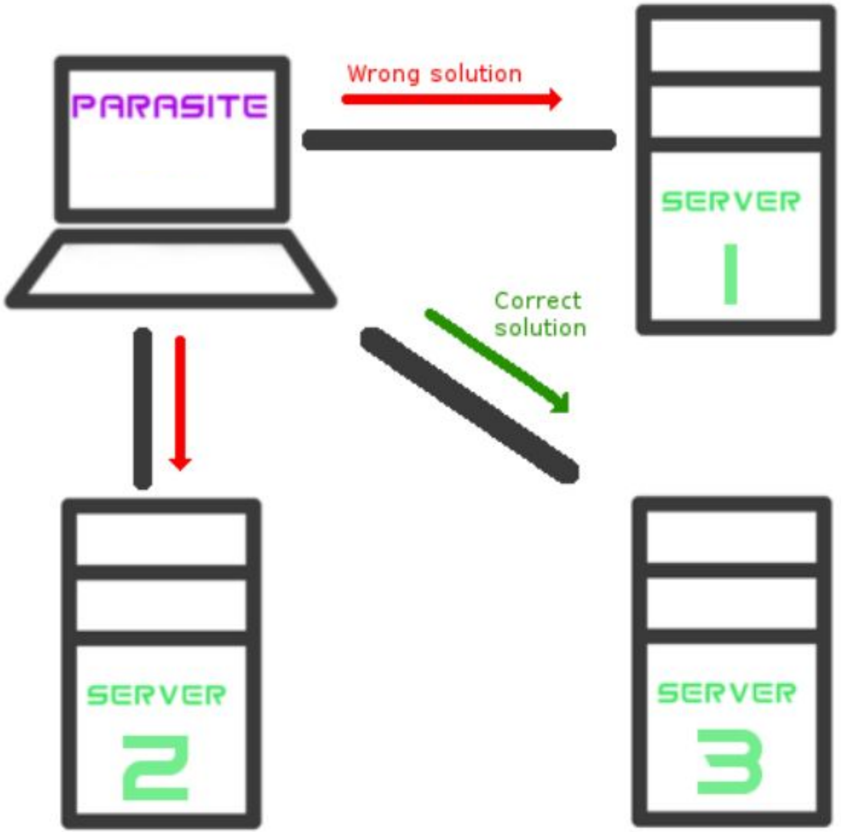
3.b.2) Invalid segment dropped by TCP

Step 3b: Compute (negative answer)

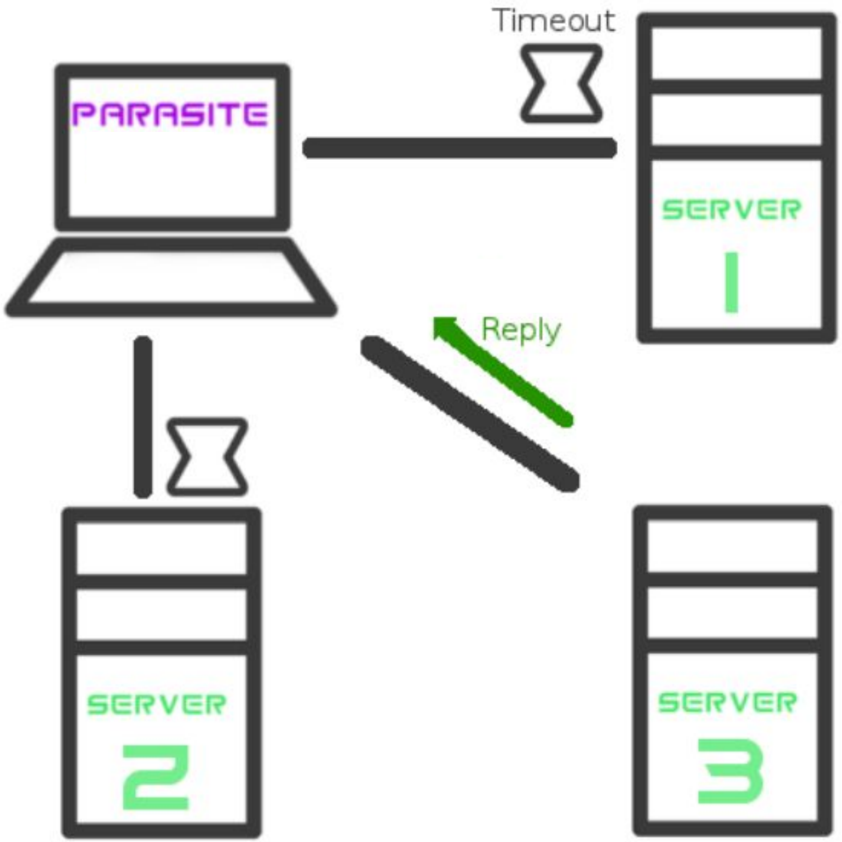


3.b.3) Parasite times out,
interprets that as negative answer

Tests are sent...



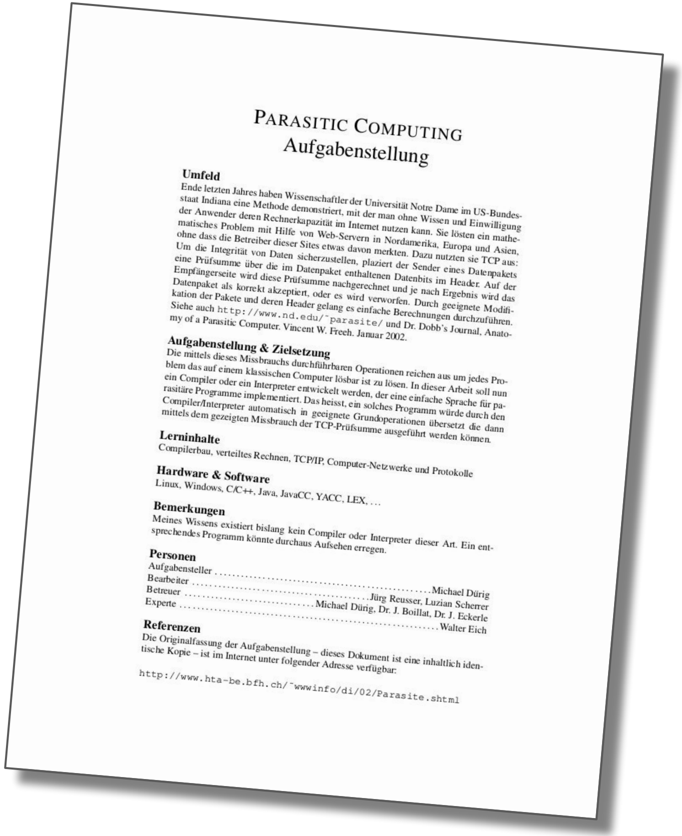
Only valid solutions are answered



One year later...

...students of the University of Applied Sciences, Bern, Switzerland, extended this concept into a **programmable virtual machine**

Any program can be compiled for this virtual machine and **solved using other's** public servers



PARASITIC COMPUTING
Aufgabenstellung

Umfeld

Ende letzten Jahres haben Wissenschaftler der Universität Notre Dame im US-Bundesstaat Indiana eine Methode demonstriert, mit der man ohne Wissen und Einwilligung der Anwender deren Rechnerkapazität im Internet nutzen kann. Sie lösten ein mathematisches Problem mit Hilfe von Web-Servern in Nordamerika, Europa und Asien, ohne dass die Betreiber dieser Sites etwas davon merkten. Dazu nutzten sie TCP aus: Um die Integrität von Daten sicherzustellen, plaziert der Sender eines Datenpakets eine Prüfsumme über die im Datenpaket enthaltenen Datenbits im Header. Auf der Empfängerseite wird diese Prüfsumme nachgerechnet und je nach Ergebnis wird das Datenpaket als korrekt akzeptiert, oder es wird verworfen. Durch geeignete Modifikation der Pakete und deren Header gelang es einfache Berechnungen durchzuführen. Siehe auch `http://www.nd.edu/~parasite/` und Dr. Dobb's Journal, Anatomy of a Parasitic Computer. Vincent W. Freeh. Januar 2002.

Aufgabenstellung & Zielsetzung

Die mittels dieses Missbrauchs durchführbaren Operationen reichen aus um jedes Problem das auf einem klassischen Computer lösbar ist zu lösen. In dieser Arbeit soll nun ein Compiler oder ein Interpreter entwickelt werden, der eine einfache Sprache für parasitäre Programme implementiert. Das heisst, ein solches Programm würde durch den Compiler/Interpreter automatisch in geeignete Grundoperationen übersetzt die dann mittels dem gezeigten Missbrauch der TCP-Prüfsumme ausgeführt werden können.

Lerninhalte

Compilerbau, verteiltes Rechnen, TCP/IP, Computer-Netzwerke und Protokolle

Hardware & Software

Linux, Windows, C/C++, Java, JavaCC, YACC, LEX, ...

Bemerkungen

Meines Wissens existiert bislang kein Compiler oder Interpreter dieser Art. Ein entsprechendes Programm könnte durchaus Aufsehen erregen.

Personen

Aufgabensteller .... Michael Dürig
Bearbeiter .... Jürg Reusser, Luzian Scherrer
Betreuer .... Michael Dürig, Dr. J. Boillat, Dr. J. Eckerle
Experte .... Walter Eich

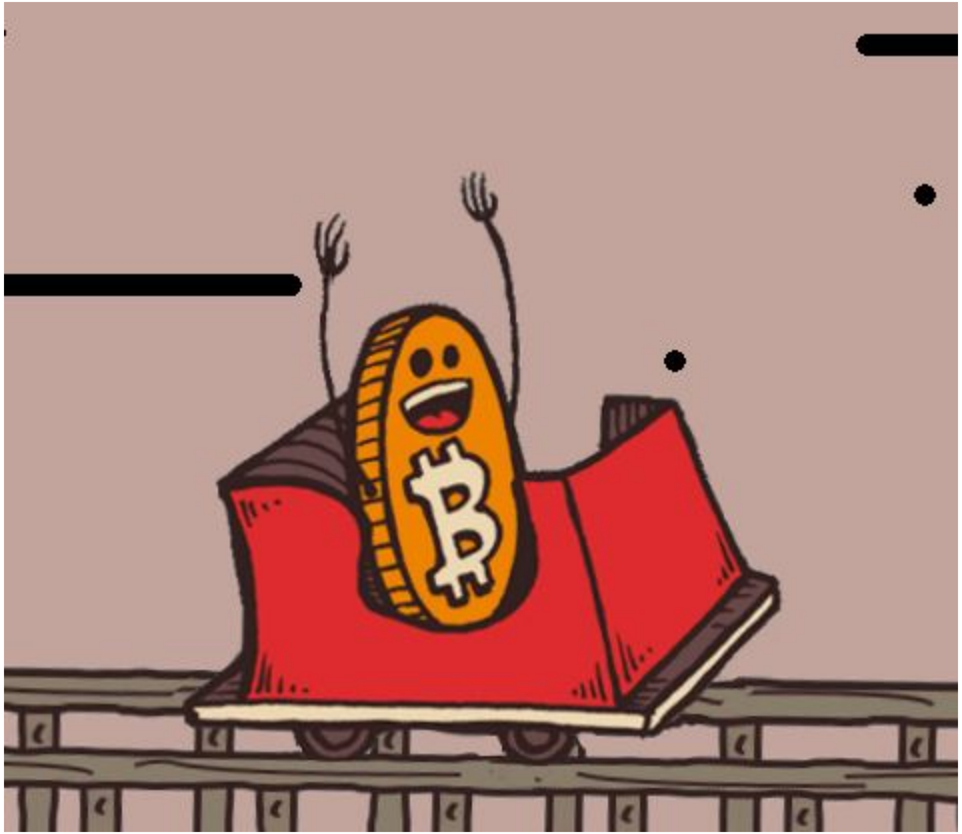
Referenzen

Die Originalfassung der Aufgabenstellung – dieses Dokument ist eine inhaltlich identische Kopie – ist im Internet unter folgender Adresse verfügbar:

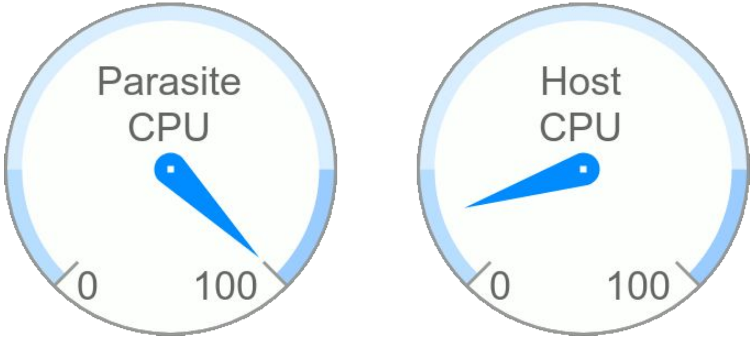
`http://www.hta-be.bfh.ch/~wwwinfo/di/02/Parasite.shtml`

Did I hear free CPU?!

Let's mine bitcoins with the
free resources of the Interwebs!



Not so fast!



The cost of encoding the problem into TCP packets is way higher than the benefits of using the host's CPU

Parasitic Computing seems **impractical** but has some **nice features**

Logic Protection

The host can't discover what the parasite is computing

- No algorithm is sent from the parasite to the host
- The parasite uses the host for simple calculations

Host Readiness

Hosts' resources are instantly available

- No need to install special software
- Parasite is capable of encoding the problem into the host's "language"

Host Resources

Host resources are available to the parasite

- Computers are more than CPUs
  - Hardware
    - Memory
    - Storage
    - GPU
  - Networking
  - Data

Can we make it practical?

The authors suggest that as one moves **up the application stack**, there might come a point where there is a **computational gain** to the parasite.

Host as Virtual Machines

TCP Checksum "Virtual Machine"

- Just one "instruction"
- Only access to CPU

Our Dream "Virtual Machine"

- High-level "instructions"
- Access to more resources
  - Hardware
  - Network
  - Data

Interpreters are the perfect target for **powerful** Parasite Computing

Logic Protection

The host can't discover what the parasite is computing

- Making use of *eval()* we can execute small pieces of code

Host Readiness

Hosts' resources are instantly available

- Every system has at least one interpreter

Host Resources

Host resources are available to the parasite

- Interpreted languages make it easy to access any kind of host resources
  - Hardware
  - Networking
  - Data

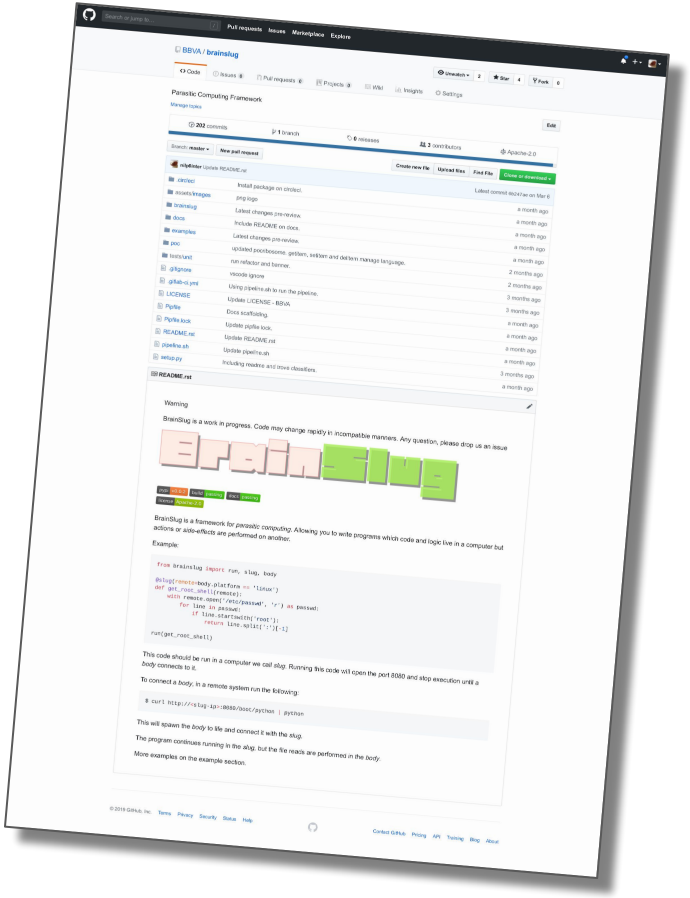
BrainSlug

A Framework for Parasitic Computing

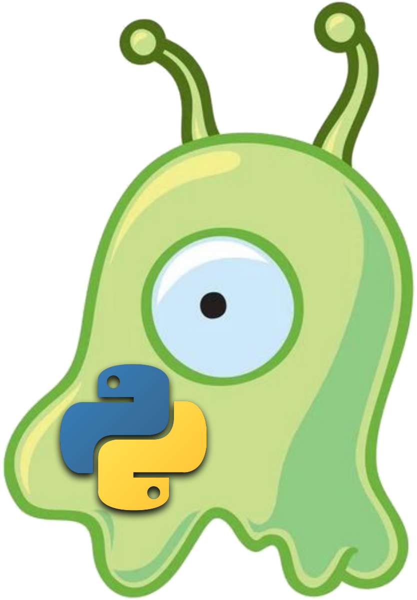
BrainSlug

A FOSS Parasitic Computing framework for Python.

It allows users to write **normal-looking Python programs** that use resources from external **interpreters**.

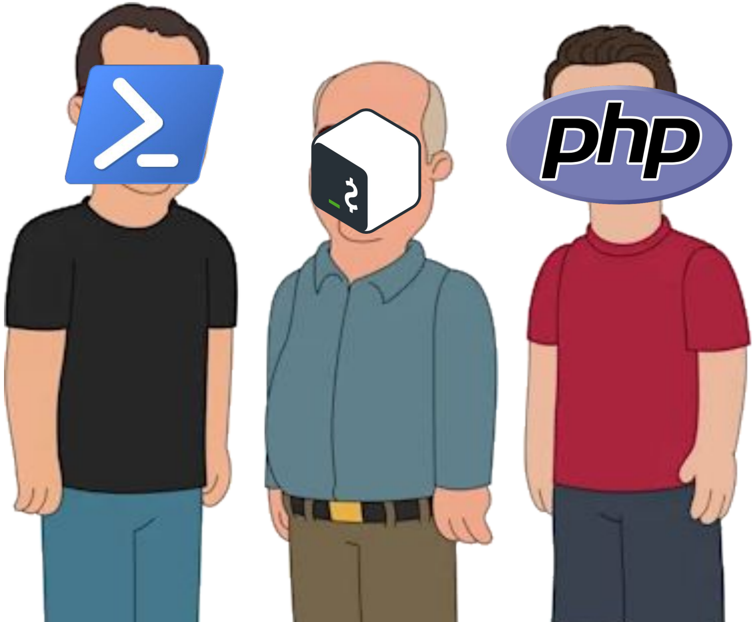


Slug [Parasite]



- Control the program flow
- Translate between Python and remote interpreters
- Serve body's initial code

Zombie Body [Host]



A small script in charge of communicating with the Slug

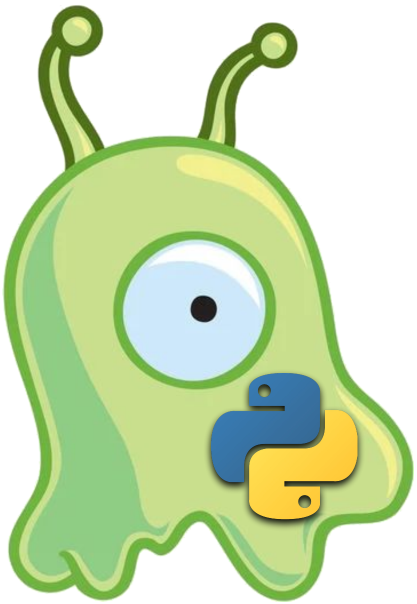
10 Download code via HTTP
20 *eval()* it
30 Send result back, getting more code
40 *GOTO* 20

Brainslug "Hello world!"

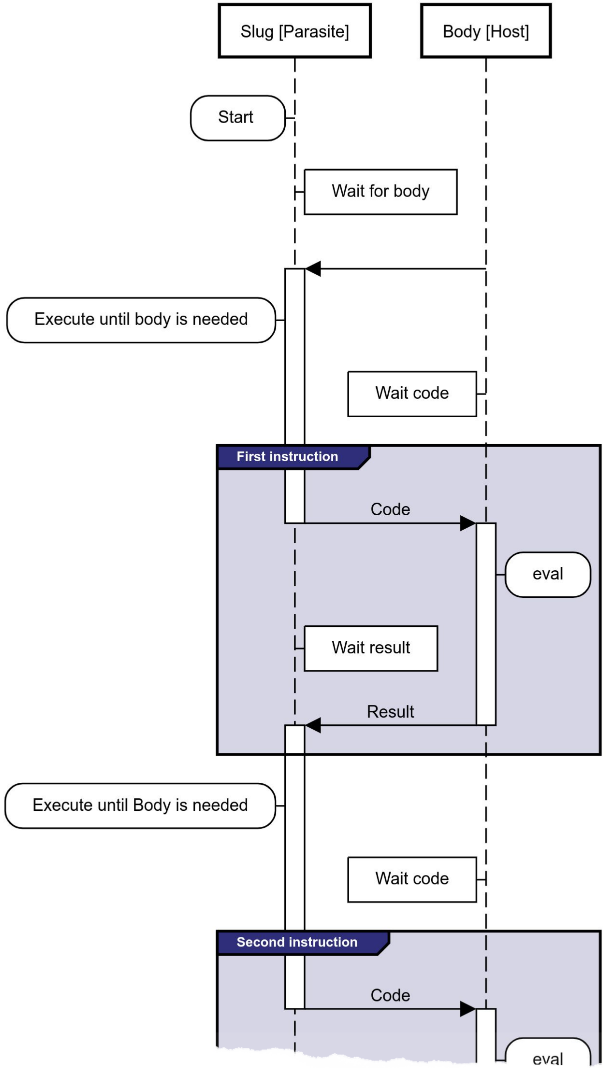
```python
from brainslug import run, slug, body

@slug(remote=body)
def helloworld(remote):
    remote.print("Hello world!")

if __name__ == '__main__':
    run(helloworld)
```

```
$> curl http://<slugip>/boot/bash | bash
```

Communication Model



Slug [Parasite]



Zombie Body [Host]

DEMO #1

Parasitic Remote Desktop for Windows

- Slug
  - Web Interface for the client
  - Ask the body for screenshots continually
  - Forward any mouse/keyboard interaction to the body
- Zombie Body
  - Powershell is all you need!
    - Capable of taking screenshots
    - Moving and clicking the mouse

DEMO TIME!

Parasitic Remote Desktop for Windows

DEMO #2

Parasitic Remote Browser

- Slug
  - The same exact Python code from last DEMO
- Body
  - Loads html2canvas on demand
  - Render screenshots in the browser

DEMO TIME!

Parasitic Remote Browser


WHAT SORCERY
IS THIS?

BrainSlug is not magic

Work has to be done in order to translate between Python and other languages

Creating a New Zombie Body (Bootstrap)?

Declare a ***boot*** function that returns the body source code for the new language

```python
from brainslug import ribosome
from brainslug.ribosome import define
from shlex import quote as escape

bash = ribosome.root("bash")

@define(bash.boot)
def _(remote, url, **kwargs):
    return f'''
      RES=""
      while true;
      do
        RES=$(curl -X POST --data-binary="$RES" {escape(url)});
      done
    '''
```

Adding Functionality

New functionality is added through ***ribosomes (translator)***

1. Responsible of encoding/decoding from/to the body
2. Use the low-level __eval__
3. May use already other body functions

```python
@define(bash.print)
def _(remote, text):
    remote.__eval__(f"echo {escape(text)}")
```

In Summary...

Use BrainSlug if you need:

1. **Logic Protection.** All the program logic is in the Slug.
2. **Host Readiness.** Leverage existing remote interpreters to avoid deploys.
3. **Host Resources.** Use remote resources occasionally.

Contribute!

New **bodies** and **ribosomes** can be distributed as Python packages.

github.com/BBVA/brainslug

QUESTION TIME!

Thank you!

github.com/BBVA/brainslug