Parasitic Computing
Complex systems student presentation
– Davide Berardi –
Parasitism is a non-mutual symbiotic relationship between species, where one species, the parasite, benefits at the expense of the other, the host.

– Wikipedia
We need to make difference between a parasitic behaviour and others similar behaviours.
We need to make difference between a parasitic behaviour and others similar behaviours.

- **Parasitism**: Only one end have benefits but it’s transparent or semi-transparent to the other.
  
  Nature: Flea/Taenia ...
  
  CS: Parasite Program (This presentation will explain this point!)
We need to make difference between a parasitic behaviour and others similar behaviours.

- **Parasitism**: Only one end have benefits but it’s transparent or semi-transparent to the other.  
  Nature: Flea/Taenia ...  
  CS: Parasite Program (This presentation will explain this point!)

- **Symbiontism**: Both ends help together and have benefits.  
  Nature: Anemone & Clownfish/Crocodile & Plover...  
  CS: GridComputing
Parasitism vs All

We need to make difference between a parasitic behaviour and others similar behaviours.

- **Parasitism**: Only one end have benefits but it’s transparent or semi-transparent to the other.
  - Nature: Flea/Taenia ...
  - CS: Parasite Program (This presentation will explain this point!)

- **Symbiontism**: Both ends help together and have benefits.
  - Nature: Anemone & Clownfish/Crocodile & Plover...
  - CS: GridComputing

- **Parasitoidism**: Only one end have benefits and consume the other one to the death.
  - Nature: Viruses/Xenomorph (Alien)...
  - CS: Viruses
We can "learn" from the parasites and implement a similar behaviour in a program?
That would be useful?
Parasitic Computing Implementation: Scenario

Let’s imagine the following scenario:

We have the bad task to compute a big SAT-3 (or some equivalent NP-complete) problem. Let's imagine the following scenario:

We have access to some servers, but we don't have any (legal) way to communicate with them without counting ICMP (ping).

We don't want to alarm a SysAdmin or harm/DOS the servers...

Wait...ICMP??
Parasitic Computing Implementation: Scenario

Let’s imagine the following scenario:

- We have the bad task to compute a big SAT-3 (or some equivalent NP-complete) problem.

\[ (x_0 \land x_1 \land x_2) \lor (\neg x_0...) \ldots \]
Let’s imagine the following scenario:

- We have the bad task to compute a big *SAT-3* (or some equivalent *NP-complete*) problem.

\[(x_0 \land x_1 \land x_2) \lor (\neg x_0 \ldots)\ldots\]

- We have access to some servers, but we don’t have any (legal) way to communicate with them without counting *ICMP* (ping).
Parasitic Computing Implementation: Scenario

Let’s imagine the following scenario:

- We have the bad task to compute a big \textit{SAT-3} (or some equivalent \textit{NP-complete}) problem. 
  \[(x_0 \land x_1 \land x_2) \lor (\neg x_0 \ldots)\ldots\]

- We have access to some servers, but we don’t have any (legal) way to communicate with them without counting \textit{ICMP} (ping).

- We don’t want to alarm a SysAdmin or harm/DOS the servers...

\[\text{CAUTION}\]
\begin{center}
\includegraphics[width=0.4\textwidth]{warning.png}
\end{center}

\text{Angry Sysadmin}
Let’s imagine the following scenario:

- We have the bad task to compute a big *SAT-3* (or some equivalent *NP-complete*) problem.
  \[(x_0 \land x_1 \land x_2) \lor (\neg x_0...)\]...

- We have access to some servers, but we don’t have any (legal) way to communicate with them without counting *ICMP* (ping).

- We don’t want to alarm a SysAdmin or harm/DOS the servers...

Wait...ICMP!?
To check if the payload of a package is good, ICMP performs a checksum.

```c
uint16_t checksum(unsigned char *buf, ssize_t len)
{
    int i = 0;
    uint16_t t csum = 0;
    for (i = 0; i < len / 2; i++)
        csum += htons((uint16_t)buf[i]);
    return ~csum;
}
```

We can use the polynomial time reduction to adapt our computation to the checksum.

$\text{SAT}^3 \leq \text{P}^{m(2^n)}$ ICMPChecksum

Then we create a packet for every possible solution with the checksum set to our fake-checksum, and the payload to the result we want to get.
To check if the payload of a package is good, ICMP performs a checksum.

```c
uint16_t checksum (unsigned char *buf, ssize_t len) {
    int i = 0;
    uint16_t csum = 0;
    for(i = 0; i < len/2; i++)
        csum += htons(((uint16_t*)buf)[i]);
    return ~csum;
}
```
Parasite Computing Implementation: ICMP/TCP exploit

- To check if the payload of a package is good, ICMP performs a checksum.

```c
uint16_t checksum (unsigned char *buf, ssize_t len)
{
    int i = 0;
    uint16_t csum = 0;
    for(i = 0; i < len/2; i ++)
        csum += htons(((uint16_t*)buf)[i]);
    return ~ csum;
}
```

- We can use the polynomial time reduction to adapt our computation to the checksum.

\[
\text{SAT3} \leq^P_m (2^n)\text{ICMPChecksum}
\]
Parasite Computing Implementation: ICMP/TCP exploit

- To check if the payload of a package is good, ICMP performs a checksum.

```c
uint16_t checksum (unsigned char *buf, ssize_t len)
{
    int i = 0;
    uint16_t csum = 0;
    for(i = 0; i < len/2; i++)
        csum+= htons(((uint16_t*)buf)[i]);
    return ~csum;
}
```

- We can use the polynomial time reduction to adapt our computation to the checksum.
  \[ SAT3 \leq^P_m (2^n)ICMPChecksum \]

- Then we create a packet for every possible solution with the checksum set to our fake-checksum, and the payload to the result we want to get.
Then the host system will receive our parasite package.

- If the checksum is wrong so the kernel will drop our package and we will get a communication timeout.
- If the solution is good then the checksum is verified and the system will reply to the ICMP request.
Then the host system will receive our parasite package.

- If the checksum is wrong so the kernel will drop our package and we will get a communication timeout.
- If the solution is good then the checksum is verified and the system will reply to the ICMP request.
Oh well, let’s do a little demo!

Let’s take a look to a parasite implementation: We will calculate a little sum of a number using a raspberry pi. And then we will calculate a computation with a bit more of complexity.
A little benchmark

I have executed the parasitic program on my Wi-Fi Lan (g)

Bubblesort with one host take more than 500 seconds (and fails...) when in simulation takes only 0.63 seconds! Fibonacci takes 8 seconds in simulation...
A little benchmark

I have executed the parasitic program on my Wi-Fi Lan (g)

Bubblesort with one host take more than 500 seconds (and fails...) when in simulation takes only 0.63 seconds! Fibonacci takes 8 seconds in simulation...
Obviously that implementation it’s only a proof of concept so it have little pros and a lot of cons:

+ Parasitic harmless programs are possible!
+ That technically legal.
+ That really hard to stop.

- If a packet get a real burst error then we get false positives/negatives.
- If the kernel/router drop the IP packet we get false negatives (1210).
- Consumes more power to build all packages than compute the direct solution. (that amplified by the bad implementation of the vm!)
Pro and Cons

Obviously that implementation it’s only a proof of concept so it have little pros and a lot of cons:

+ Parasitic harmless programs are possible!
Pro and Cons

Obviously that implementation it’s only a proof of concept so it have little pros and a lot of cons:

- Parasitic harmless programs are possible!
- That technically legal.

- If a packet get a real burst error then we get false positives/negatives.
- If the kernel/router drop the IP packet we get false negatives (121).
- Consumes more power to build all packages than compute the direct solution. (that amplified by the bad implementation of the vm!)
Obviously that implementation it’s only a proof of concept so it have little pros and a lot of cons:

+ Parasitic harmless programs are possible!
+ That technically legal.
+ That really hard to stop.
Obviously that implementation it’s only a proof of concept so it have little pros and a lot of cons:

+ Parasitic harmless programs are possible!
+ That technically legal.
+ That really hard to stop.
- If a packet get a real burst error then we get false positive/negatives.
Pro and Cons

Obviously that implementation it’s only a proof of concept so it have little pros and a lot of cons:

+ Parasitic harmless programs are possible!
+ That technically legal.
+ That really hard to stop.
  - If a packet get a real burst error then we get false positive/negatives.
  - If the kernel/router drop the IP packet we get false negatives ($\frac{1}{2^{10}}$).
Pro and Cons

Obviously that implementation it’s only a proof of concept so it have little pros and a lot of cons:

+ Parasitic harmless programs are possible!
+ That technically legal.
+ That really hard to stop.
- If a packet get a real burst error then we get false positive/negatives.
- If the kernel/router drop the IP packet we get false negatives \( \left( \frac{1}{2^{10}} \right) \).
- Consumes more power to build all packages than compute the direct solution. (that amplified by the bad implementation of the vm!)
Conclusions

Thank you for your attention!

And happy Paras-hacking!

http://www.szene.ch/parasit/code/index.html
http://www3.nd.edu/~parasite/