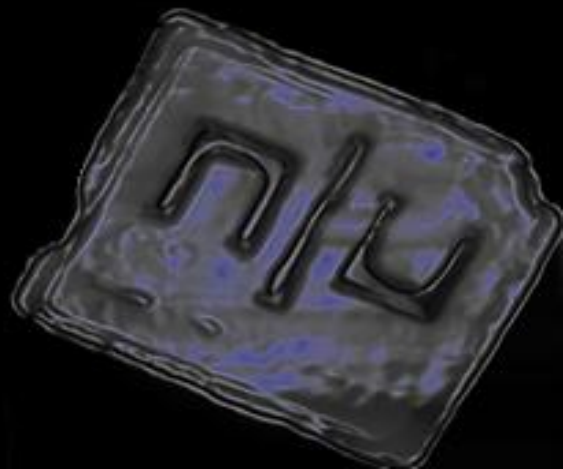
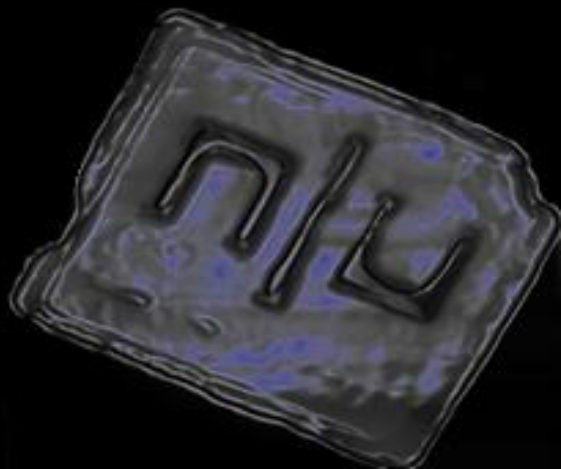
Wireplay: An approach to **almost** Blind Fuzzing

- Abhisek Datta



Agenda

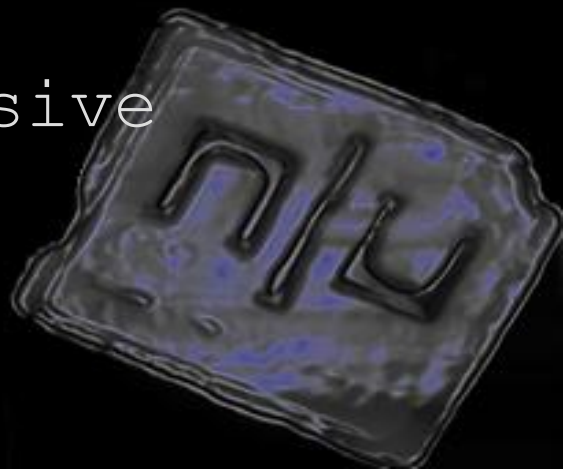
- Little Theory about fuzzing
- Problems & Solution
- Introducing Wireplay
- Field Testing
- Wireplay Hooks



Fuzz Testing != Hacking

- Feeding random/semi-random valid/invalid data set to various input interfaces of a program and monitor for possible faults!

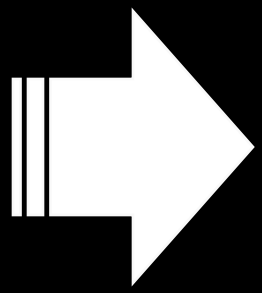
- Fuzzing does find expensive security bugs!

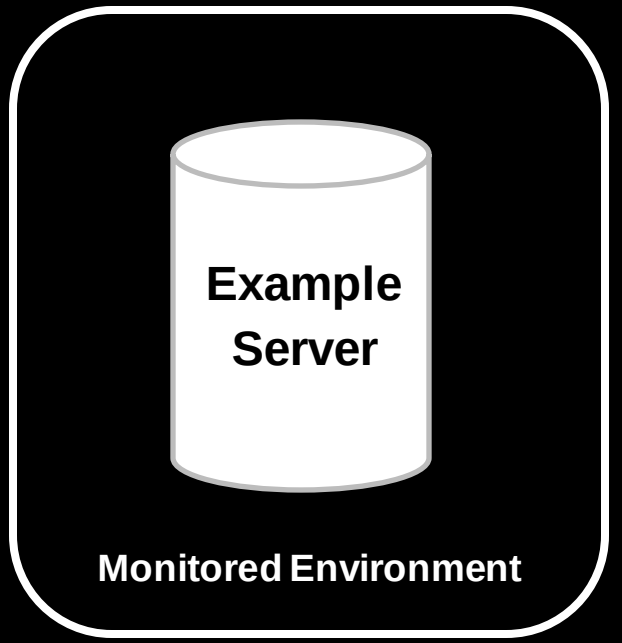


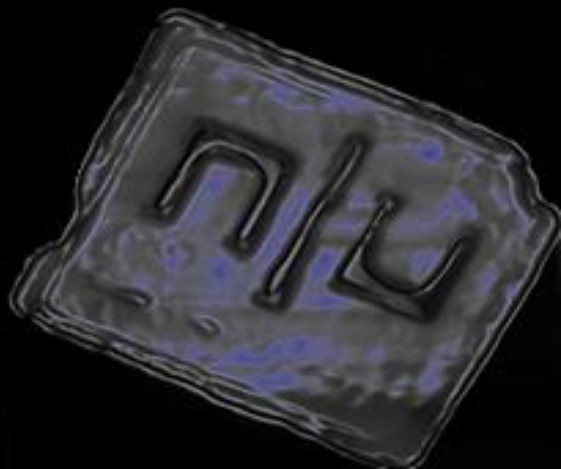
Fuzz Testing aka. Fuzzing

SELECT **name**
FROM **users** WHERE **id** = **10**

SELECT
AAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
AAAAAAAAAAAA
FROM **users** WHERE **id** = **10**

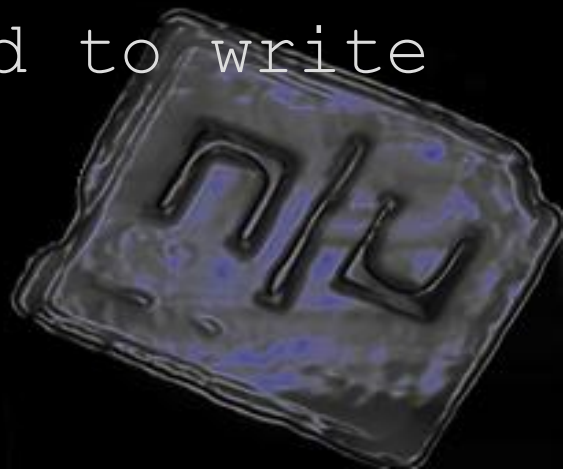






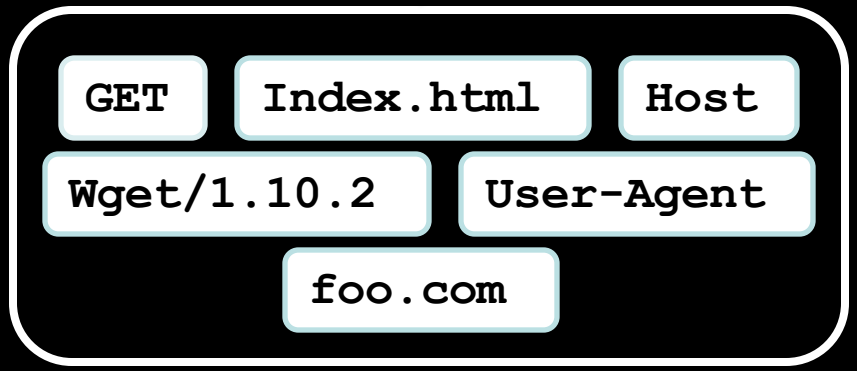
Block Based Modeling

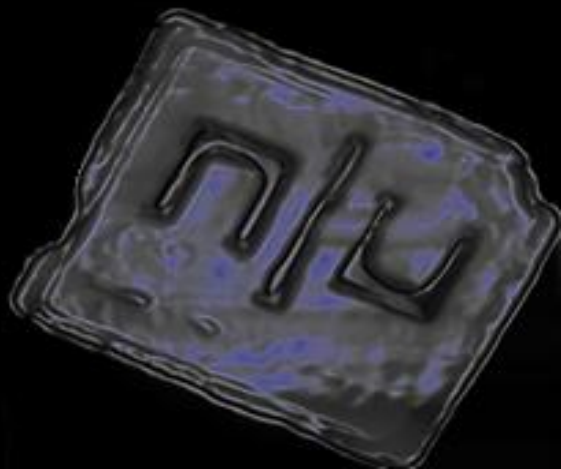
- A theoretical approach to model the problem of Fuzzing.
  - Original (valid) Input Set is tokenized (blocks) and each token is fuzzed periodically.
  - Better approach than blind fuzzing, however you need to write a LOT of code!
    - SPIKE, Peach, Sully etc.



Block Based Fuzzing

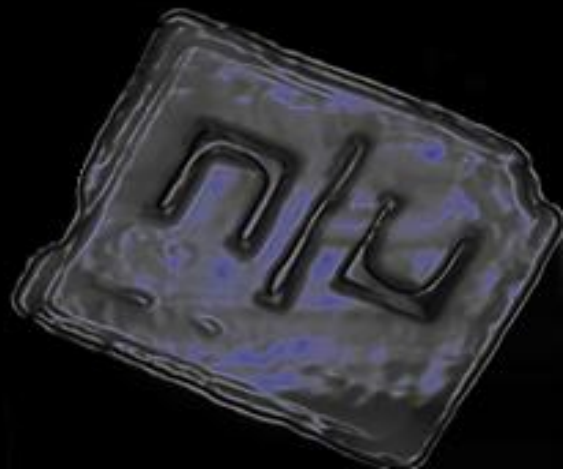
GET /index.html HTTP/1.1
Host: foo.com
User-Agent: wget/1.10.2





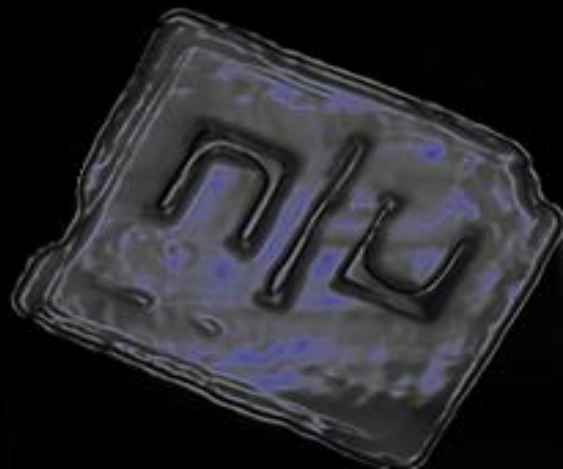
Block Based Fuzzing

```
for-each-token
    data.replace(token, get_random())
    target.send(data)
end
```



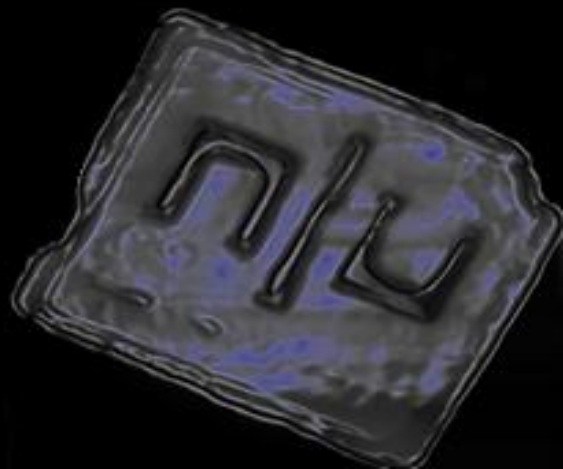
Block Based Fuzzing

GET <BIG-RANDOM-STRING> HTTP/1.1
Host: foo.com
User-Agent: wget/1.10.2



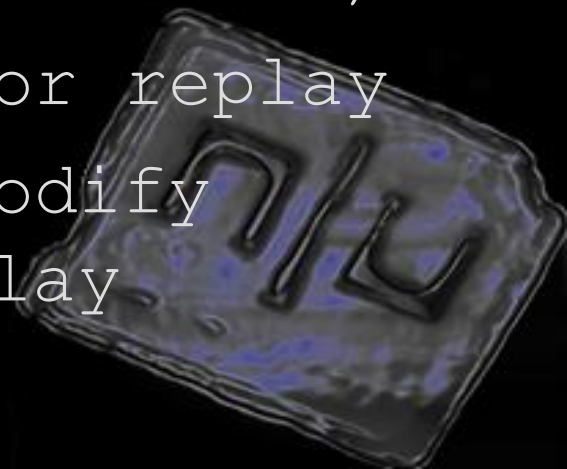
Block Based Fuzzing: Problems

- Tokenization needs knowledge of protocol.
  - Lots of protocols.
  - Proprietary protocols.
  - Time Consuming.
  - ETC.

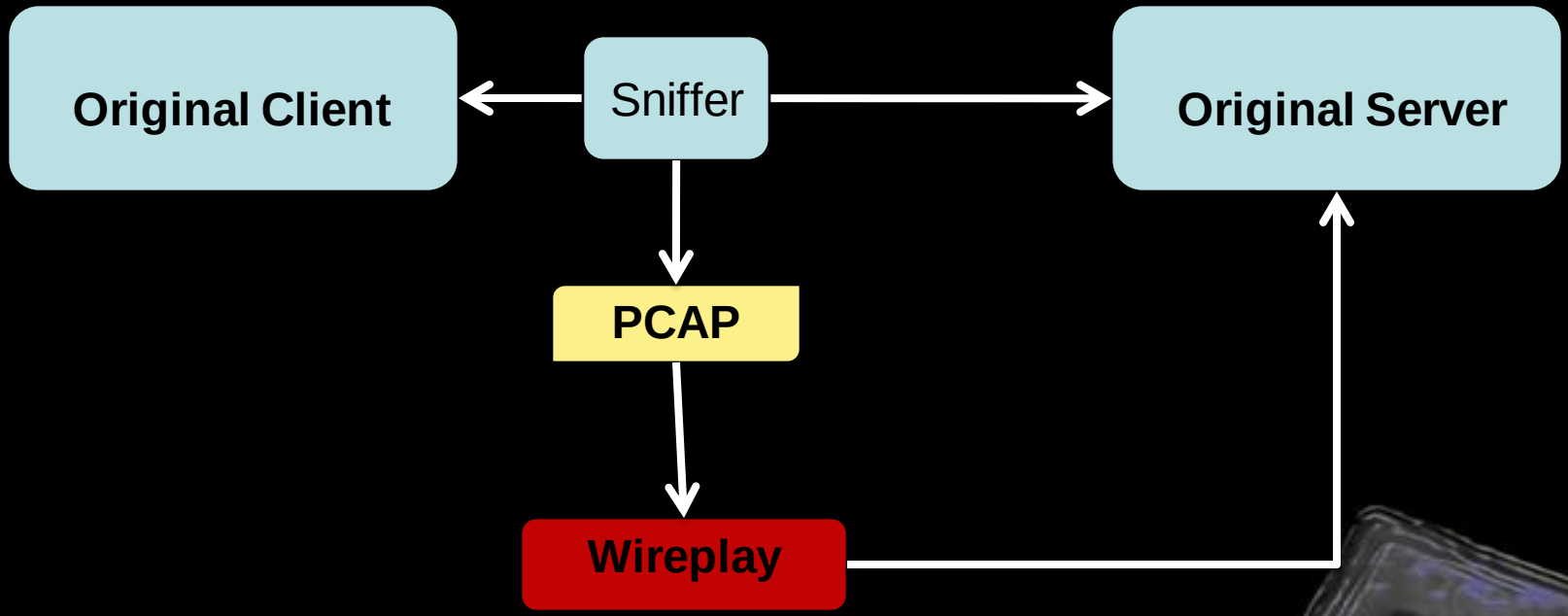


Introducing Wireplay

- Minimalist approach to replay TCP Sessions with modifications as required.
  - Use your valid client to connect to the server.
  - Capture the packets (Wireshark?)
  - Feed them to Wireplay for replay
  - Use Wireplay hooks to modify original packets and replay



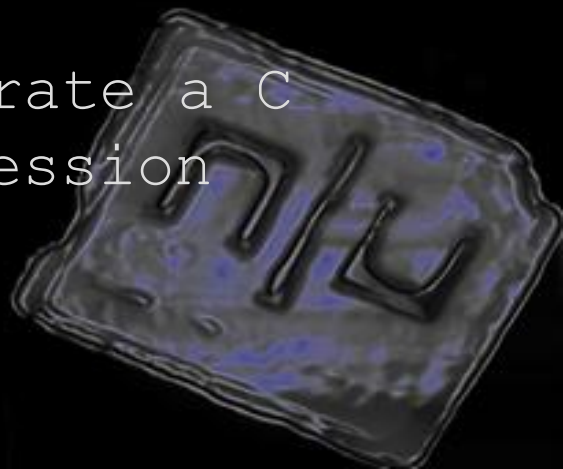
Wireplay: Functional Flow



http://code.google.com/p/wireplay

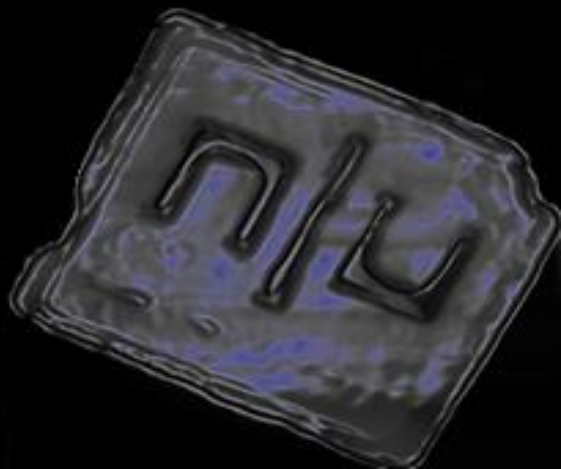
Wireplay Features

- TCP Stream replay
  - TCP Session reconstruction via. modified libnids (bug fixes)
- Plugin Subsystem
  - Ruby Interpreter Embedded as Plugin
  - Supports Packet Mangling hooks written in Ruby
  - CGEN: A ruby plugin to generate a C program to reproduce a TCP Session



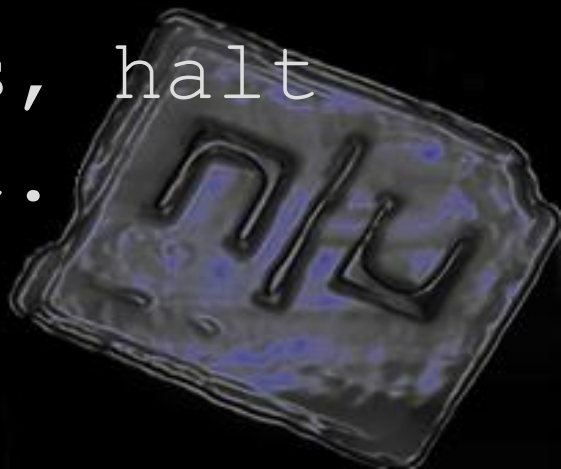
Wireplay: Basic Usage

```
bash$ wireplay -r client \
        -t 172.16.0.1 \
        -p 80
        -F pcap/http.pcap \
        -K # optional
```

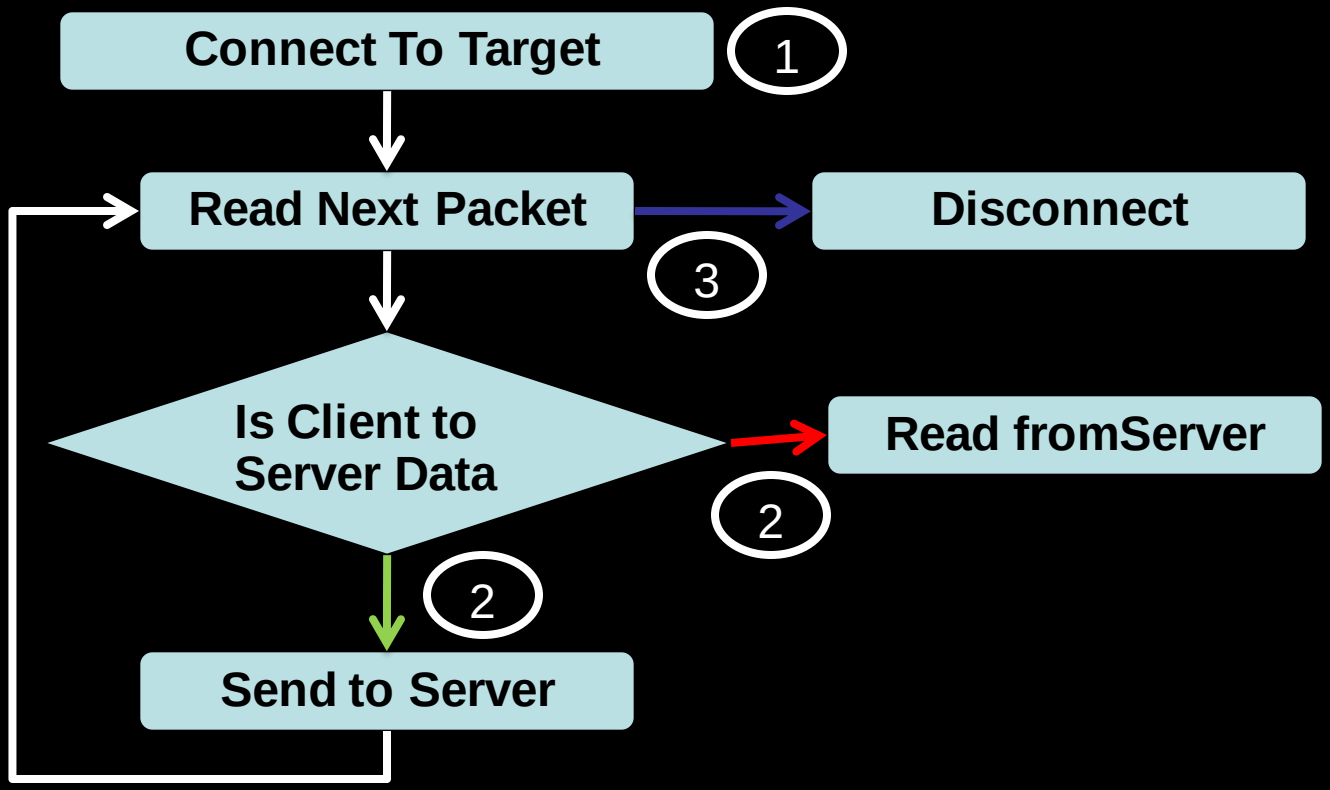


Wireplay: Fuzzing

- Hook Subsystem for arbitrary data manipulation.
  - Embedded Ruby Interpreter and API set for writing packet manipulation hooks in Ruby
- Misc. features to repeat fuzz sessions, ignore errors, halt on connection fault etc.

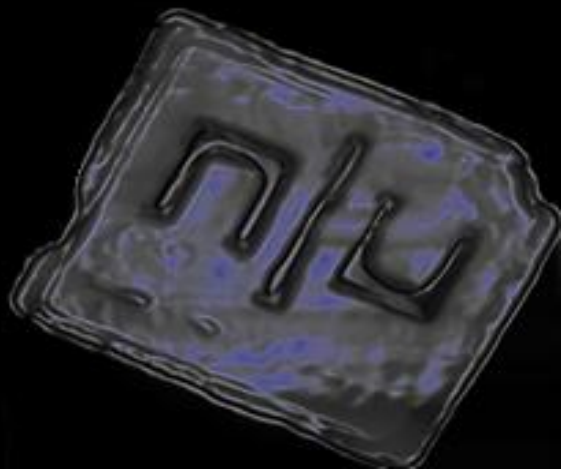


Wireplay: Hook System



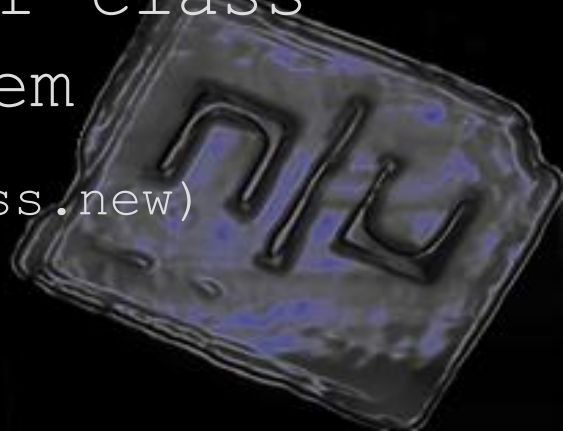
Events

1. On Start
2. On Data
3. On Stop
4. On Error



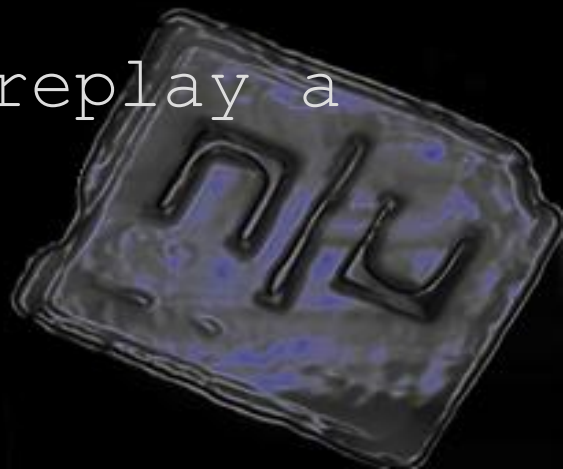
Wireplay: Packet Hook in Ruby

- Define your arbitrary class with the following methods:
  - **on_start(pkt_desc)**
  - **on_stop(pkt_desc)**
  - **on_data(pkt_desc, direction, data)**
  - **on_error(pkt_desc, code)**
- Register an object of your class with Wireplay Hook Subsystem
  - `Wireplay::Hooks.register(YourClass.new)`

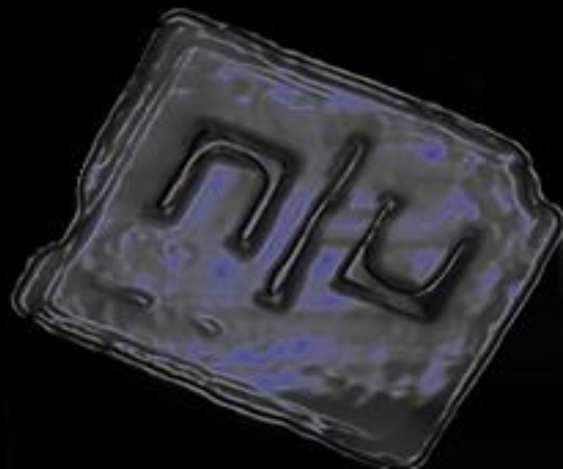


Wireplay: Sample Hooks

- Blind Byte Alternation (blind.rb)
  - Alters each byte from the original payload with single or multiple bytes for fuzzing.
- CGEN (cgen.rb)
  - Generates C program to replay a TCP session. Use for PoC generation.



Wireplay: **Demo**



Thank You.. ☺

svn co **http://wireplay.googlecode.com/svn/trunk** wireplay

http://code.google.com/p/wireplay/