GRAYHAT

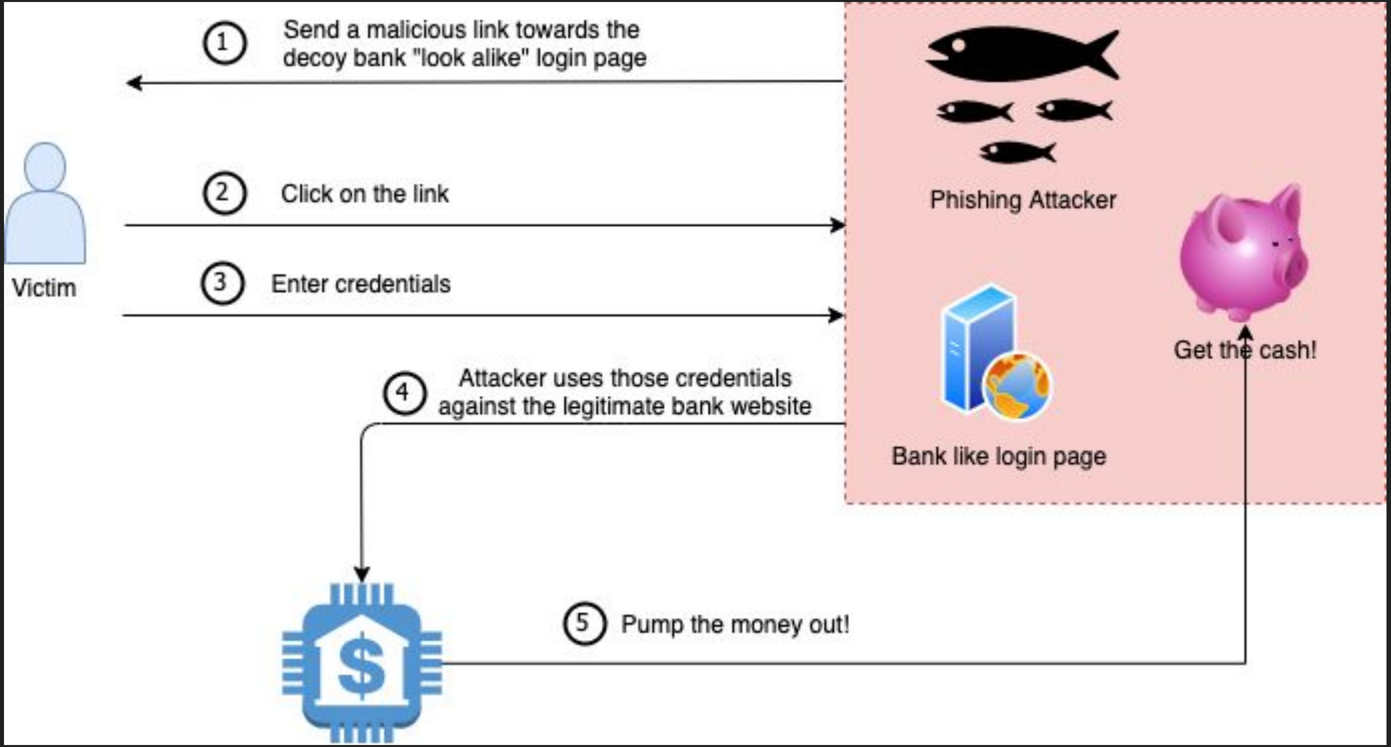
Attacking Networks with pCraft

Sebastien Tricaud
Security Engineering Director at Devo Inc
@tricaud

#whoami

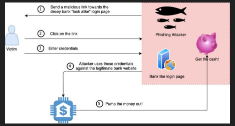
- Security Research Director at Devo
- MISP contributor
- Lead developer of SightingDB (https://github.com/stricaud/sightingdb)
- Lead developer of Faup (https://github.com/stricaud/faup)
- Honeynet Project, former CTO and current board member
- Worked on Linux PAM, Prelude IDS etc.

Have you ever seen this?



1 Send a malicious link towards the decoy bank "look alike" login page
Phishing Attacker
Victim
2 Click on the link
3 Enter credentials
Get the cash!
4 Attacker uses those credentials against the legitimate bank website
Bank like login page
5 Pump the money out!

Diagrams are insufficient


① Send a malicious link towards the decoy bank "look alike" login page
Phishing Attacker
Victim
② Click on the link
③ Enter credentials
Get the cash!
Bank like login page
④ Attacker uses those credentials against the legitimate bank website
⑤ Pump the money out!

- They summarize an attack for the general public
- You are not the general public, you want to scratch the surface!
- You REALLY want to understand the attack, details are hidden
- You search the web and ask your friends if they have a PCAP
- Would you have detected it with your IDS, SIEMs etc.?

Creating a pcap about this Phishing attack

- Clicking on a link
  - Do a DNS request to get the IP address of the Host
  - HTTP Session:
    - Send TCP Three Way Handshake
    - Send the HTTP Request
    - Get an HTTP Response
    - Terminate
- Pcraft was built to help you do this
  - https://github.com/devoinc/pcraft

Installation

Required tools

- Pcraft (https://github.com/devoinc/pcraft)
- TCPreplay https://tcpreplay.appneta.com/
- Suricata https://suricata-ids.org/

Docker

```
$ docker pull sightingdb/pcraft

$ docker run --name pcraft -d sightingdb/pcraft
```

Your first pcap

Let's create our first pcap

```
ami_version 1

action DnsRequest {
     $domain = "grayhat.co"
     exec DNSConnection
}
```

Let's run to create our first pcap

```
ami_version 1

action DnsRequest {
      $domain = "grayhat.co"
      exec DNSConnection
}
```

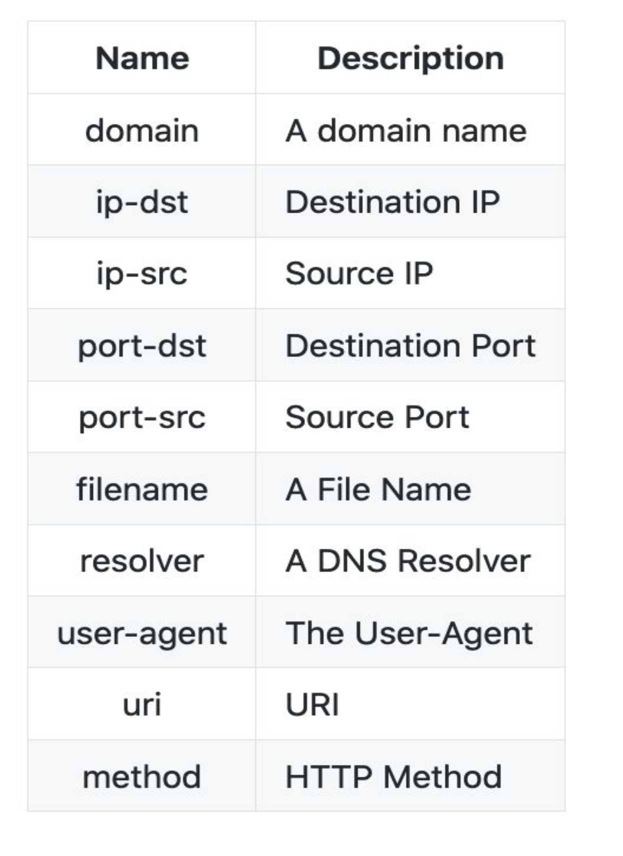
```
$ ./pcrafter dns.ami dns.pcap

$ tshark -r dns.pcap

     1   0.000000 192.168.214.149 → 1.1.1.1      DNS 77  Standard query 0x0000 A grayhat.co

     2   0.000824    1.1.1.1 → 192.168.214.149 DNS 110  Standard query response 0x0000 A grayhat.co A
10.252.148.119
```

Pcraft Engine

Taxonomy

In pcraft a Taxonomy is just to have all the Plugins agreeing with each other on how you label something such as a "source IP address". It shall be "ip-src".

It is stored in docs/taxonomy.md and will evolve anytime we need something new. However, the ones that are set will never change.



Name	Description
domain	A domain name
ip-dst	Destination IP
ip-src	Source IP
port-dst	Destination Port
port-src	Source Port
filename	A File Name
resolver	A DNS Resolver
user-agent	The User-Agent
uri	URI
method	HTTP Method

Adding the ip-dst in our DNS transaction

```
ami_version 1

action DnsRequest {
     $domain = "grayhat.co"
     $resolver = "8.8.8.8"
     $ip-dst = "141.193.213.20"
     exec DNSConnection
}
```

Creating our domain outside of the DNS part (one way)

```
ami_version 1

action GenerateDomain {
     exec GenerateNewDomain
}

action DnsRequest {
     $resolver = "8.8.8.8"
     $ip-dst = "141.193.213.20"
     exec DNSConnection
}
```

- Thank to Taxonomy, as long as we have a variable set with the "domain" key, it can be used instead.
- We use the GenerateNewDomain plugin which will create a valid non-existing domain, using "domain" as the way to communicate output to the other plugins
- We can remove the domain variable from the DnsRequest step

Creating our domain outside of the DNS part (another way)

```
ami_version 1

$domain = "grayhat.co"

action DnsRequest {
      $resolver = "8.8.8.8"
      $ip-dst = "141.193.213.20"
      exec DNSConnection
}
```

- Instead of using the GenerateNewDomain plugin, we set the domain variable ourselves
- We make the $domain variable available for the global scope. A Variable can be reused either automatically if a Plugin depends on it, or using $variable as an argument to a parameter

Taxonomy in plugins

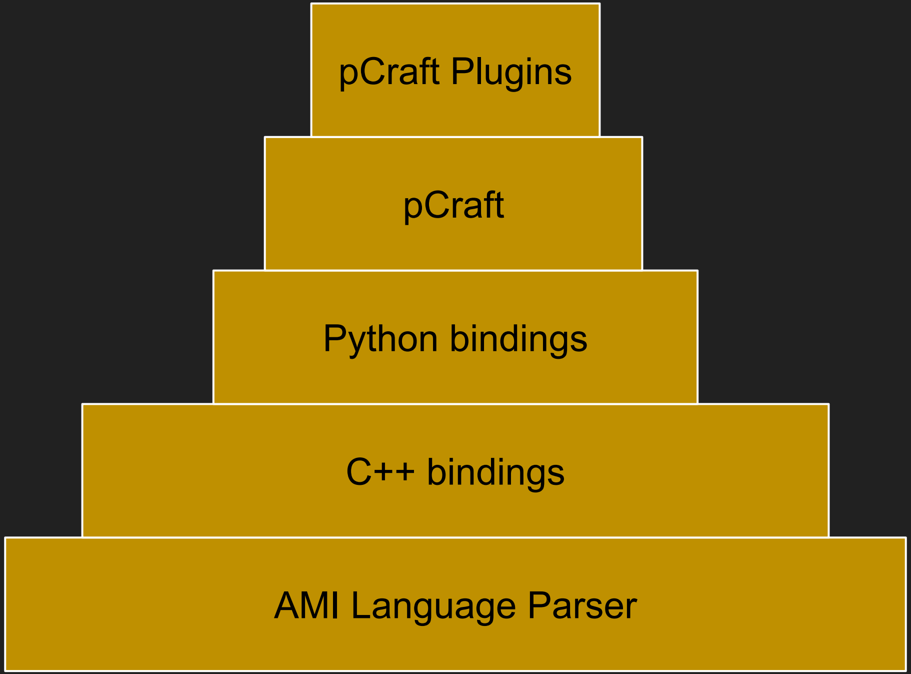
- Each action type is defined by a Plugin
- Plugins are given a plugin context (state that remain in the entire flow)
- They define the required variables: required = ["ip-dst", "domain"]; They will not work if they are not set
- The get variables from that context: self.getvar("ip-dst")

Available Plugins

- TcpRst
  - Add a TCP-RST exchange
- HTTPConnection
  - Create an HTTP connection
- GenerateNewDomain
  - Create a new valid non existing domain
- DNSConnection
  - Create a DNS query and reply
- Ping
  - Add an ICMP request and reply
- PcapImport
  - Append data from another PCAP

- HostnameFromIP
  - Generate a consistent fake hostname from an IP address
- FakeNames
  - Generate 3 variables: firstname, lastname and email
- Suricata
  - Import a Suricata rule to create a packet that will trigger that rule
- <YOURS SOON>

pCraft, under the hood!



pCraft Plugins

pCraft

Python bindings

C++ bindings

AMI Language Parser

AMI? A New Language?

- We originally used YAML, but had variables and loops, which made YAML unfit for our need
- Most action-driven languages are Json or XML (or Pascal Functions in the PLC world!)
- Wanted a simple language to describe pCraft actions
- Make the Attack Scenario easy to understand, deal with the implementation details later on

AMI in 3 slides: 1/3

```
action Whatever {
     $variable = "string"
     $verbatim = """my "verbatim" string"""
     exec MyActionToExecute
}
```

AMI in 3 slides: 2/3

```
action Whatever {
     $variable = "string"
     $verbatim = """my "verbatim" string"""
     exec MyActionToExecute
}

action AnotherOne {
     $variable = csv("file.csv", 1, "field",
has_header=true)
     exec MyOtherActionToExecute
}
```

AMI in 3 slides: 3/3

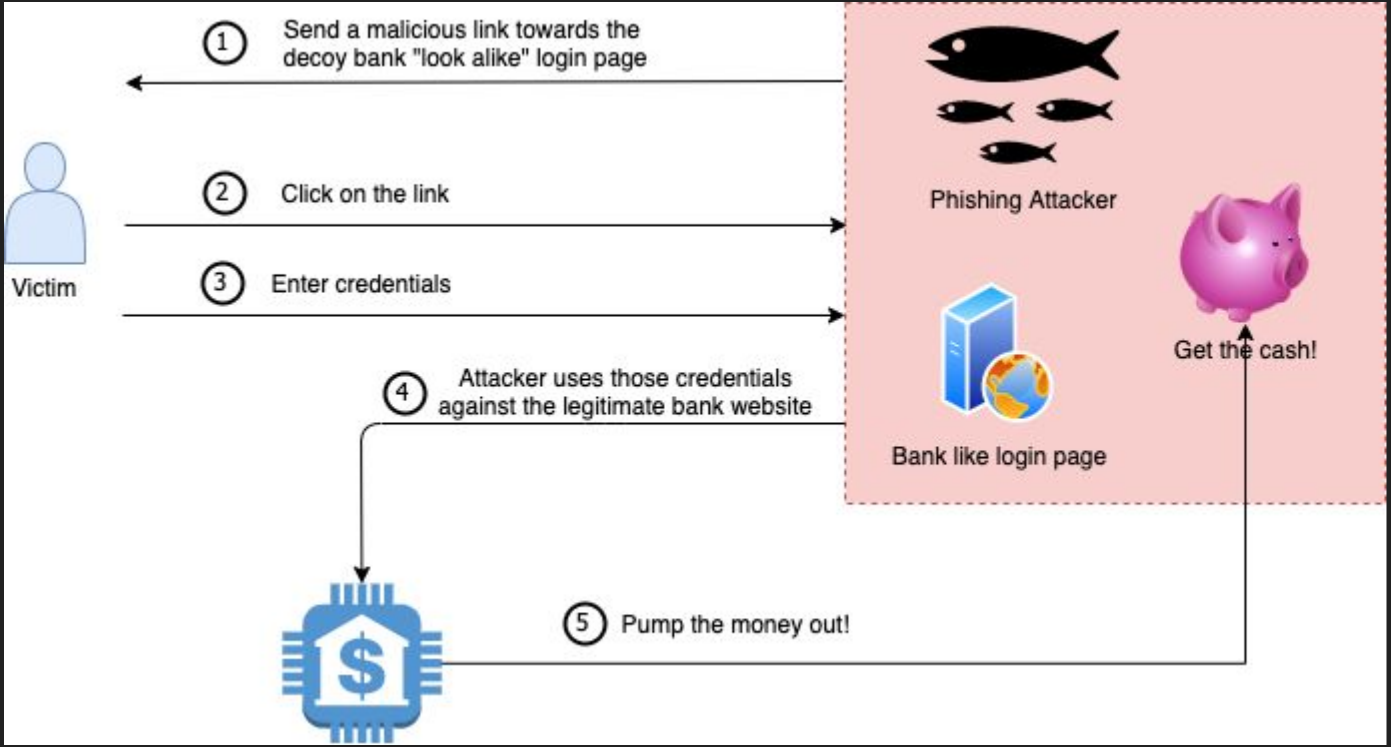
```
repeat 2 as $index { # We repeat 3 times!
  action Whatever {
       $variable = "string"
       $verbatim = """my "verbatim" string"""
       exec MyActionToExecute
  }

  action AnotherOne {
       $variable = csv("file.csv", $index,
"field", has_header=true)
       exec MyOtherActionToExecute
  }
}
```

Using AMI with Python

```
#!/usr/bin/env python3
import pyami
import sys

if __name__ == "__main__":
     if len(sys.argv) < 2:
          print("Syntax: %s script.ami" % sys.argv[0])
          sys.exit(1)

     amifile = sys.argv[1]
     amictx = pyami.Ami()
     amictx.Parse(amifile)
     actions = amictx.GetActions()

     for a in actions:
          print("name:%s action to execute:%s" % (a.Name(), a.Exec()))
```

Your first scenario

Remember this?



1 Send a malicious link towards the decoy bank "look alike" login page
2 Click on the link
3 Enter credentials
4 Attacker uses those credentials against the legitimate bank website
5 Pump the money out!
Victim
Phishing Attacker
Bank like login page
Get the cash!

Starting

```
ami_version 1

$victimip = "192.168.0.55"
$fakebankip = "185.199.108.153"
```

- We specify the AMI version to 1
- We set the victim IP and the Fake Bank IP so it will be easier to know what they are later on

The user clicks on the link

```
action GenerateADomain {
    exec GenerateNewDomain
}

action DnsRequest {
    $ip-src = $victimip
    $ip-dst = $fakebankip
    exec DNSConnection
}
```

- We call the GenerateNewDomain to create a "domain" variable. It is a hidden variable, created by the Plugin
- We jump to the dns request creation (DnsRequest)
- We call the DNSConnection plugin using the victim ip which will resolve to the fake bank IP

Generate the HTTP Post

```
action PostData {
      exec HTTPConnection
      $method = "POST"
      $client-content-type  =
"application/x-www-form-urlencoded"
      $client-content  =
"login=Alfred.Wallace@example.com&password=q
werty1234"
}
```

- We call the HTTPConnection plugin to create the POST form we want which includes the login and password for our victim

Adding a loop?

```
repeat 4 as $index {
      action PostData {
           exec HTTPConnection
           $method = "POST"
           $client-content-type  =
"application/x-www-form-urlencoded"
           $client-content  =
"login=Alfred.Wallace@example.com&password=q
werty1234"
      }
}
```

- We add the repeat block around the actions we want to repeat
- 4 is how many times we repeat
- $index describes the iterator index variable

Writing a Plug-in

Where do plugins live?

- If you have the docker image, they are in:
    - /usr/local/lib/python3.8/dist-packages/pcraft/plugins/
    - We are running as root in this docker image, so you can play with the directory
- If you ran a git clone on https://github.com/devoinc/pcraft they are in pcraft/plugins
- If you ran pip3 install pcraft, you have to find out
    - One trick: `$ pcrafter /bin/ls foo`
        - Forget the error, look at the first line which dumps the loaded plugins with their directories
- They must be dropped in the plugin/ directory so it would be loaded at startup time with pcraft

Plugin structure

```python
from pcraft.PluginsContext import PluginsContext

class PCraftPlugin(PluginsContext):
    name = "MyOwnStuff"

    def __init__(self, app, session, plugins_data):
        super().__init__(app, session, plugins_data)

    def run(self, ami, action):
        print("Hello, from MyOwnStuff")

        return self.plugins_data
```

Plugin structure

1

```
from pcraft.PluginsContext import PluginsContext
```

2

```
class PCraftPlugin(PluginsContext):
    name = "MyOwnStuff"
```

3

```
    def __init__(self, app, session, plugins_data):
        super().__init__(app, session, plugins_data)
```

4

```
    def run(self, ami, action):
        print("Hello, from MyOwnStuff" )

        return None, self.plugins_data
```

1. We import the PluginsContext
2. We create a class "PCraftPlugin" and we call it with the same name that will be used for the file
3. Initialization with Instancing from PluginsContext
4. Called when we run:
    a. Your code
    b. Return the plugin_data (the pcap we write / variables..)

Passing Variables

```
def run(self, ami, action):

    print("The domain variable:" + self.getvar("domain"))
    self.setvar("ip-src", "192.168.0.0")

    self.plugins_data
```

- Simply use self.setvar() and self.getvar()

Packet Manipulation

- We are writing a pcap
    - We craft a packet using Scapy
    - We append that packet to the array which is used in the end to write the whole pcap
- Import Scapy in your Plugin:
    - `from scapy.all import *`

Packet Manipulation

- We are writing a pcap
  - We craft a packet using Scapy
  - We append that packet to the array which is used in the end to write the whole pcap
- Import Scapy in your Plugin:
  - `from scapy.all import *`
- Mangle your ICMP packet the Scapy way:

```
echo_request = Ether() / IP(src=self.getvar["ip-src"], dst=str(individual_ip)) /
ICMP(type="echo-request")
```

Packet Manipulation

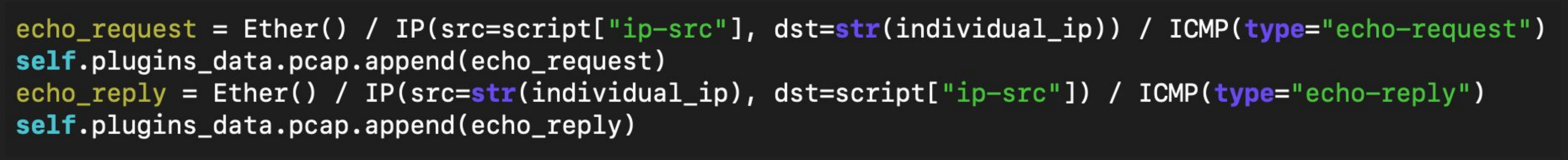
- We are writing a pcap
  - We craft a packet using Scapy
  - We append that packet to the array which is used in the end to write the whole pcap
- Import Scapy in your Plugin:
  - `from scapy.all import *`
- Mangle your ICMP packet the Scapy way:

```
echo_request = Ether() / IP(src=self.getvar("ip-src"), dst=str(individual_ip)) /
ICMP(type="echo-request")
```

- Append your packet to the global array

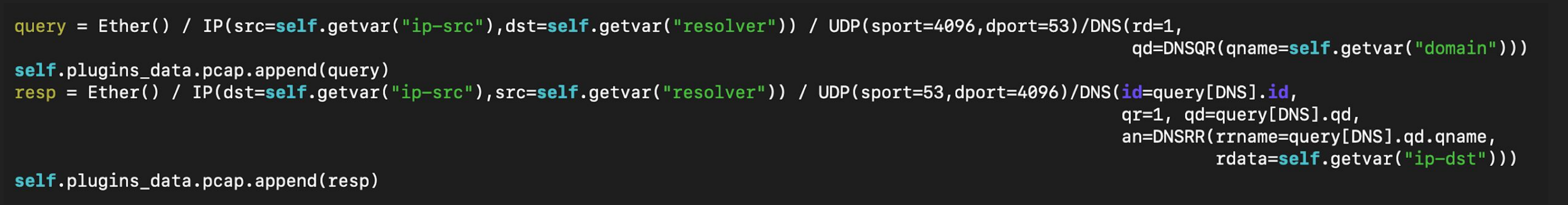
```
self.plugins_data.pcap.append(echo_request)
```

Example, packet writing with ICMP



```
echo_request = Ether() / IP(src=script["ip-src"], dst=str(individual_ip)) / ICMP(type="echo-request")
self.plugins_data.pcap.append(echo_request)
echo_reply = Ether() / IP(src=str(individual_ip), dst=script["ip-src"]) / ICMP(type="echo-reply")
self.plugins_data.pcap.append(echo_reply)
```

DNS Packet writing



```
query = Ether() / IP(src=self.getvar("ip-src"),dst=self.getvar("resolver")) / UDP(sport=4096,dport=53)/DNS(rd=1,
                                                                                                           qd=DNSQR(qname=self.getvar("domain")))
self.plugins_data.pcap.append(query)
resp = Ether() / IP(dst=self.getvar("ip-src"),src=self.getvar("resolver")) / UDP(sport=53,dport=4096)/DNS(id=query[DNS].id,
                                                                                                          qr=1, qd=query[DNS].qd,
                                                                                                          an=DNSRR(rrname=query[DNS].qd.qname,
                                                                                                                   rdata=self.getvar("ip-dst")))
self.plugins_data.pcap.append(resp)
```

TCP

- TCP requires more work, because there is a handshake, we must keep the session flow consistent etc.
- We have a util library that helps to write the handshake for us
- We have a session helper to give the proper sequence and acknowledgement number

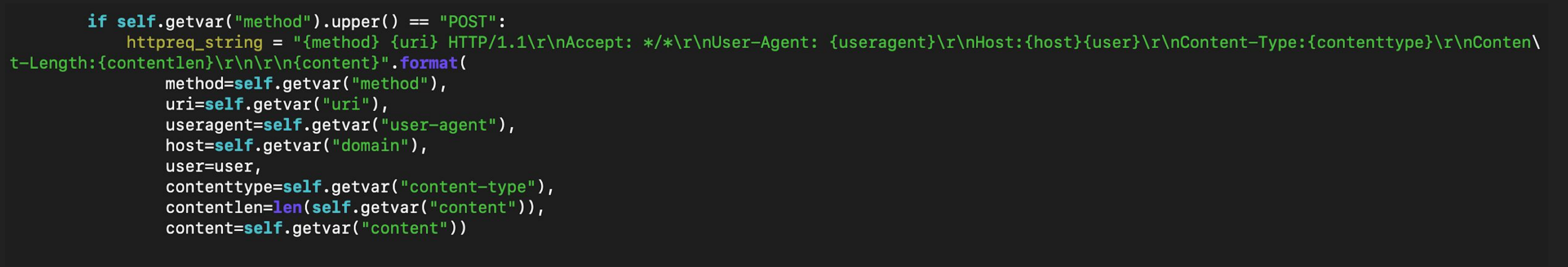
Adding a three-way handshake

```
from . import _utils as utils

utils.append_tcp_three_way_handshake(self.session,
self.plugins_data, self.getvar("port-src"))
```

Correct Sequence and Acknowledgement

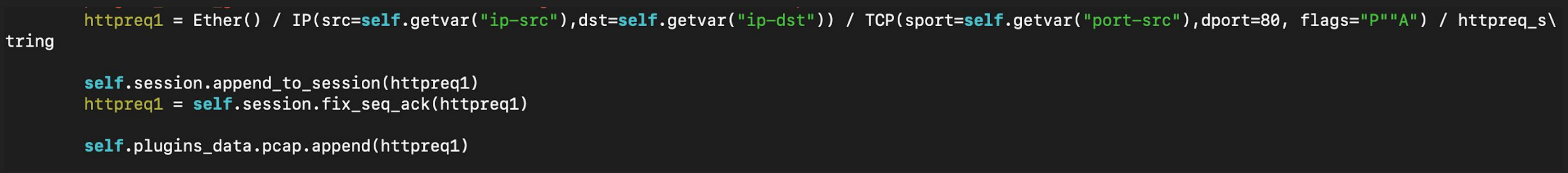
```
packet = Ether() / IP( …

self.session.append_to_session(packet)

packet = self.session.fix_seq_ack(packet)

self.plugins_data.pcap.append(packet)
```

Example from the HTTPConnection Plugin



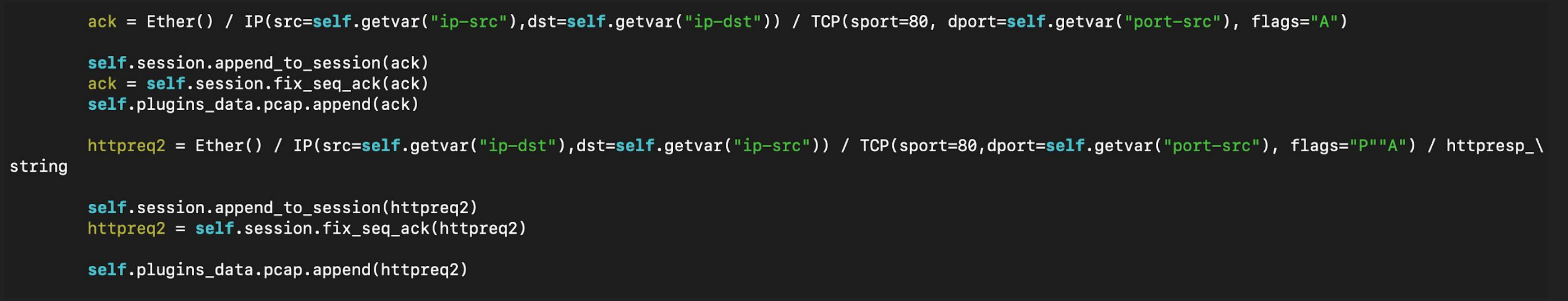
```
        if self.getvar("method").upper() == "POST":
            httpreq_string = "{method} {uri} HTTP/1.1\r\nAccept: */*\r\nUser-Agent: {useragent}\r\nHost:{host}{user}\r\nContent-Type:{contenttype}\r\nConten\
t-Length:{contentlen}\r\n\r\n{content}".format(
                method=self.getvar("method"),
                uri=self.getvar("uri"),
                useragent=self.getvar("user-agent"),
                host=self.getvar("domain"),
                user=user,
                contenttype=self.getvar("content-type"),
                contentlen=len(self.getvar("content")),
                content=self.getvar("content"))
```

Example from the HTTPConnection Plugin



```
        httpreq1 = Ether() / IP(src=self.getvar("ip-src"),dst=self.getvar("ip-dst")) / TCP(sport=self.getvar("port-src"),dport=80, flags="P""A") / httpreq_s\
tring

        self.session.append_to_session(httpreq1)
        httpreq1 = self.session.fix_seq_ack(httpreq1)

        self.plugins_data.pcap.append(httpreq1)
```

Example from the HTTPConnection Plugin



```
        ack = Ether() / IP(src=self.getvar("ip-src"),dst=self.getvar("ip-dst")) / TCP(sport=80, dport=self.getvar("port-src"), flags="A")

        self.session.append_to_session(ack)
        ack = self.session.fix_seq_ack(ack)
        self.plugins_data.pcap.append(ack)

        httpreq2 = Ether() / IP(src=self.getvar("ip-dst"),dst=self.getvar("ip-src")) / TCP(sport=80,dport=self.getvar("port-src"), flags="P""A") / httpresp_\
string

        self.session.append_to_session(httpreq2)
        httpreq2 = self.session.fix_seq_ack(httpreq2)

        self.plugins_data.pcap.append(httpreq2)
```

Advanced Scenario

Import another pcap

```
action Collection {
    exec PcapImport
    $filename = "phishing.pcap"
    field["ip"].replace("192.168.0.55" => "10.0.43.2", "185.199.108.153" =>
"172.16.38.5")
}
```

Add a Suricata rule!

```
action PlayWithSuricata {
    exec Suricata
    $EXTERNAL_NET = "192.168.0.55"
    $HTTP_SERVERS = "141.193.213.20"
    $ip-dst = "141.193.213.20"
    $domain = "grayhat.co"
    $rule = """alert http $EXTERNAL_NET any -> $HTTP_SERVERS any (msg:"ET
WEB_SERVER Possible Custom Content Type Manager WP Backdoor Access";
flow:established,to_server;http.uri;content:"/plugins/custom-content-type-manager/aut
o-update.php"; fast_pattern;
nocase;reference:url,blog.sucuri.net/2016/03/when-wordpress-plugin-goes-bad.html;
classtype:trojan-activity; sid:2022596; rev:4; metadata:created_at 2016_03_06,
updated_at 2020_06_24;)
"""
}
```

Use functions! Loops and functions!

```
repeat 2 as $index {
    $domain = csv("fromcsv.csv", $index, field="domain", has_header=true)
    action PostData {
        exec DNSConnection
    }
}
```

```
fromcsv.csv:

ipsource,domain
127.0.0.1,cezasaduzo
192.168.0.42,zizicofydi
192.168.10.20,zizicotepa
10.20.32.12,bararepim
```

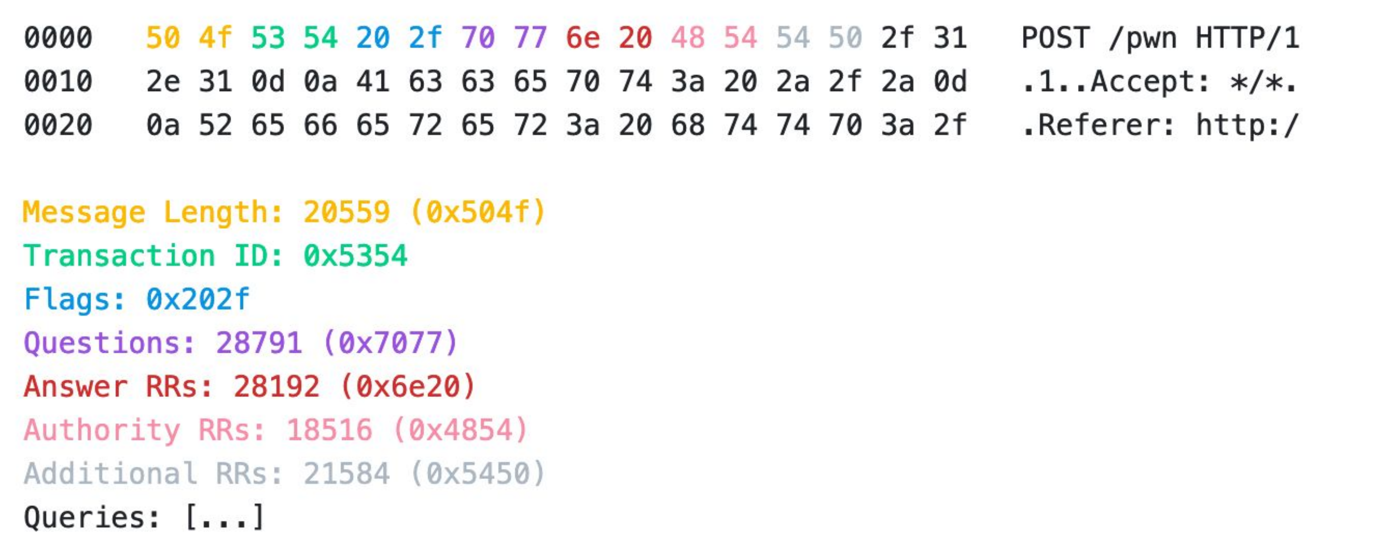
Create your attacks!

Built a Plugin or scenario?

- Please create a PR or an issue on https://github.com/devoinc/pcraft so I can add them to the repository and everybody will benefit!

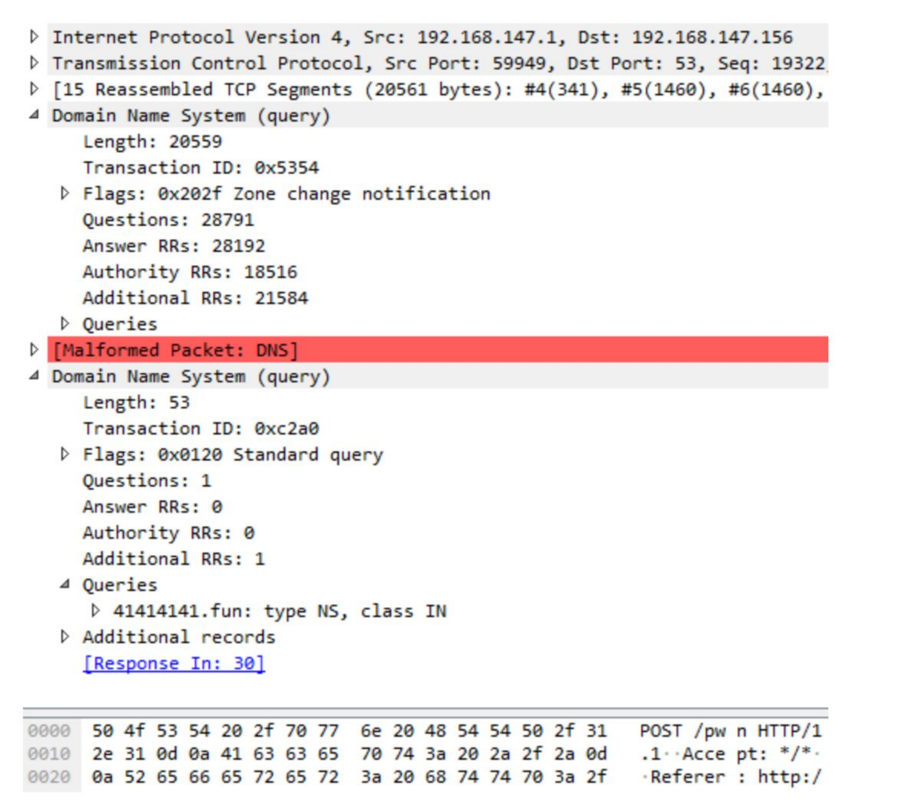
Example, Windows DNS Server vulnerability

- https://research.checkpoint.com/2020/resolving-your-way-into-domain-admin-exploiting-a-17-year-old-bug-in-windows-dns-servers/



```
0000   50 4f 53 54 20 2f 70 77 6e 20 48 54 54 50 2f 31   POST /pwn HTTP/1
0010   2e 31 0d 0a 41 63 63 65 70 74 3a 20 2a 2f 2a 0d   .1..Accept: */*.
0020   0a 52 65 66 65 72 65 72 3a 20 68 74 74 70 3a 2f   .Referer: http:/

Message Length: 20559 (0x504f)
Transaction ID: 0x5354
Flags: 0x202f
Questions: 28791 (0x7077)
Answer RRs: 28192 (0x6e20)
Authority RRs: 18516 (0x4854)
Additional RRs: 21584 (0x5450)
Queries: [...]
```

Example, Windows DNS Server vulnerability



```
▷ Internet Protocol Version 4, Src: 192.168.147.1, Dst: 192.168.147.156
▷ Transmission Control Protocol, Src Port: 59949, Dst Port: 53, Seq: 19322
▷ [15 Reassembled TCP Segments (20561 bytes): #4(341), #5(1460), #6(1460),
◢ Domain Name System (query)
    Length: 20559
    Transaction ID: 0x5354
  ▷ Flags: 0x202f Zone change notification
    Questions: 28791
    Answer RRs: 28192
    Authority RRs: 18516
    Additional RRs: 21584
  ▷ Queries
▷ [Malformed Packet: DNS]
◢ Domain Name System (query)
    Length: 53
    Transaction ID: 0xc2a0
  ▷ Flags: 0x0120 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 1
  ◢ Queries
    ▷ 41414141.fun: type NS, class IN
  ▷ Additional records
    [Response In: 30]

0000  50 4f 53 54 20 2f 70 77  6e 20 48 54 54 50 2f 31   POST /pw n HTTP/1
0010  2e 31 0d 0a 41 63 63 65  70 74 3a 20 2a 2f 2a 0d   .1··Acce pt: */*·
0020  0a 52 65 66 65 72 65 72  3a 20 68 74 74 70 3a 2f   ·Referer : http:/
```

Playing with Tcpreplay and Suricata

GRAYHAT

Create a simple Suricata rule

- In a file called "mydns.rule":

```
alert dns any any -> any any (msg:"DNS Query GrayHat"; dns_query;
content:"grayhat"; nocase; sid:20200809; rev:1;)
```

- Run Suricata:

```
suricata -S mydns.rule -i eth0
```

Add this rule into a pcraft scenario

```
start: suricata
suricata:
  _plugin: Suricata
  ip-src: 172.17.0.2
  ip-dst: 185.199.108.153
  rule: |
    alert dns any any -> any any (msg:"DNS Query GrayHat"; dns_query;
content:"grayhat"; nocase; sid:20200809; rev:1;)
  _next: done
```

Tcpreplay and monitor the Suricata log

```
tcpreplay -i eth0 suricata.pcap
Actual: 2 packets (178 bytes) sent
in 0.000880 seconds
Rated: 202272.7 Bps, 1.61 Mbps,
2272.72 pps
Flows: 2 flows, 2272.72 fps, 2 flow
packets, 0 non-flow
Statistics for network device: eth0
    Successful packets:       2
    Failed packets:           0
    Truncated packets:        0
    Retried packets (ENOBUFS): 0
    Retried packets (EAGAIN):  0
```

```
tail -f /var/log/suricata/fast.log

08/08/2020-05:35:02.696600  [**]
[1:20200809:1] DNS query alert [**]
[Classification: (null)] [Priority:
3] {PROTO:017} 172.17.0.2:4096 ->
1.1.1.1:53
```

```
\o\ \o/ /o/
```

GRAYHAT

Thank you!

Reach out to me on twitter: @tricaud
Via email: sebastien.tricaud@devo.com
Contribute! https://github.com/devoinc/pcraft