Machine Detectable Network Behavioural Commonalities for Exploits & Malware

University of Amsterdam
MSc System & Network Engineering
Research Project II

Alexandros Stavroulakis

What is this about?

Automatic generation of malicious code by the penetration testing tool, Armitage, which is a GUI of the Metasploit Framework

More specifically

When it is used by inexperienced users (hackers) and/or hobbyists

What is the problem?

A large part of ad-hoc created malware is generated using Armitage

It is possible to generate a new virus / trojan which will be hardly detectable by AV software

Why are we researching this?

To determine whether this automated generation procedure, produces code that has predictable network behaviour,

Such as packet sizes, rhythm of packets, sequence of ports, etc

If Armitage generated malware could be detected by its network behaviour characteristics, then malware detection solutions could take a major step forward

Which leads us to the Research Question

Is it possible to detect the presence of malicious software, generated by Armitage, by identifying its network behaviour?

What is the plan?

Set up a secure “victim” environment (roll-back after each trial)

I. Windows 7 SP1 Virtual Machine
II. Kali Linux Virtual Machine

Create a feature plan of malware generation using Armitage

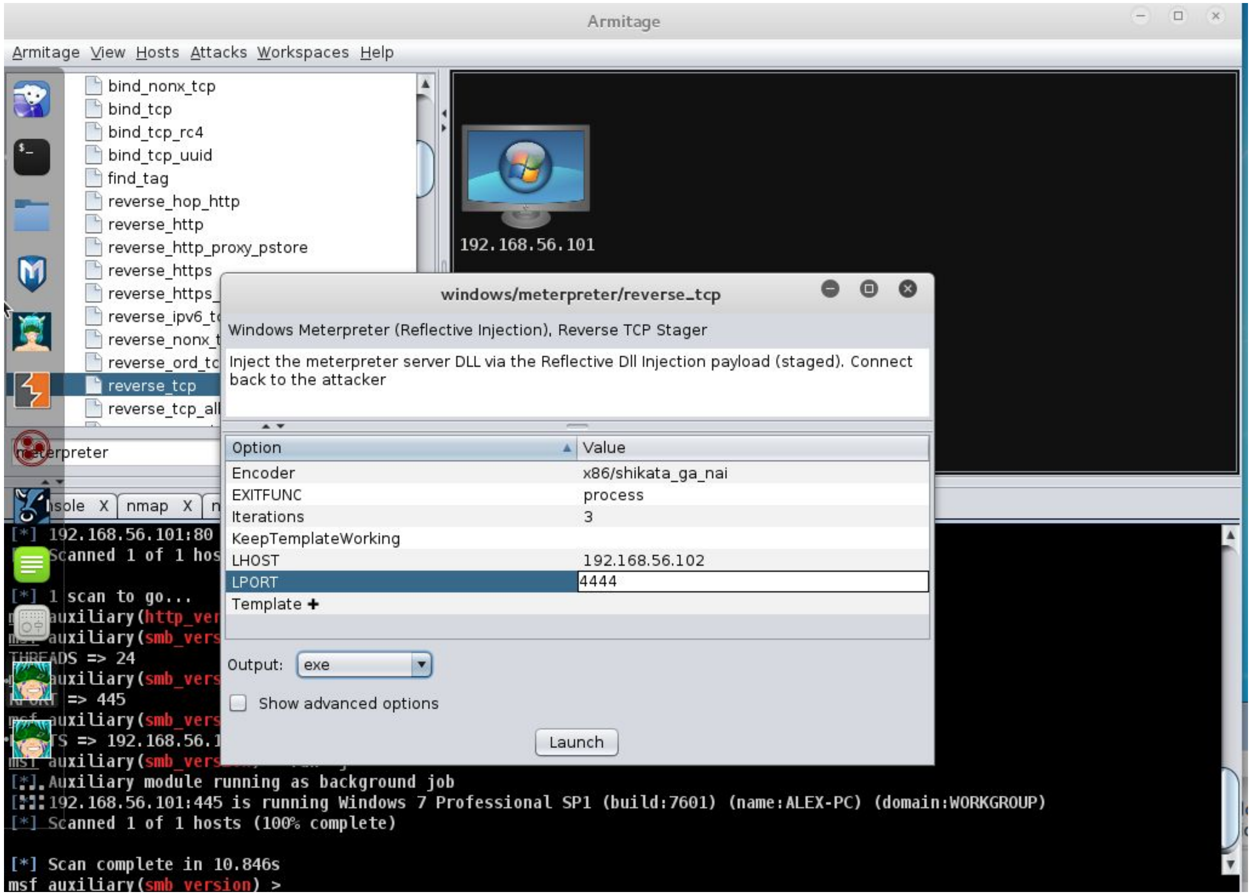
Capture and analyze traffic

How is malware generated?

Malware == Metasploit Payloads

LHOST and LPORT are set for the attacking side

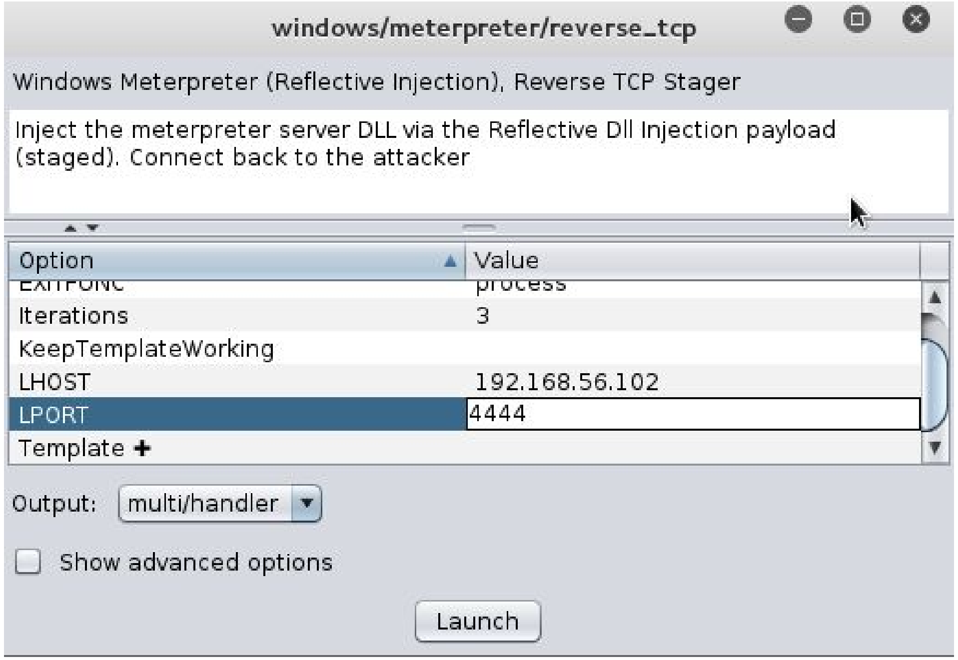
Figure out a way to infect the victim with executable



How is malware generated?

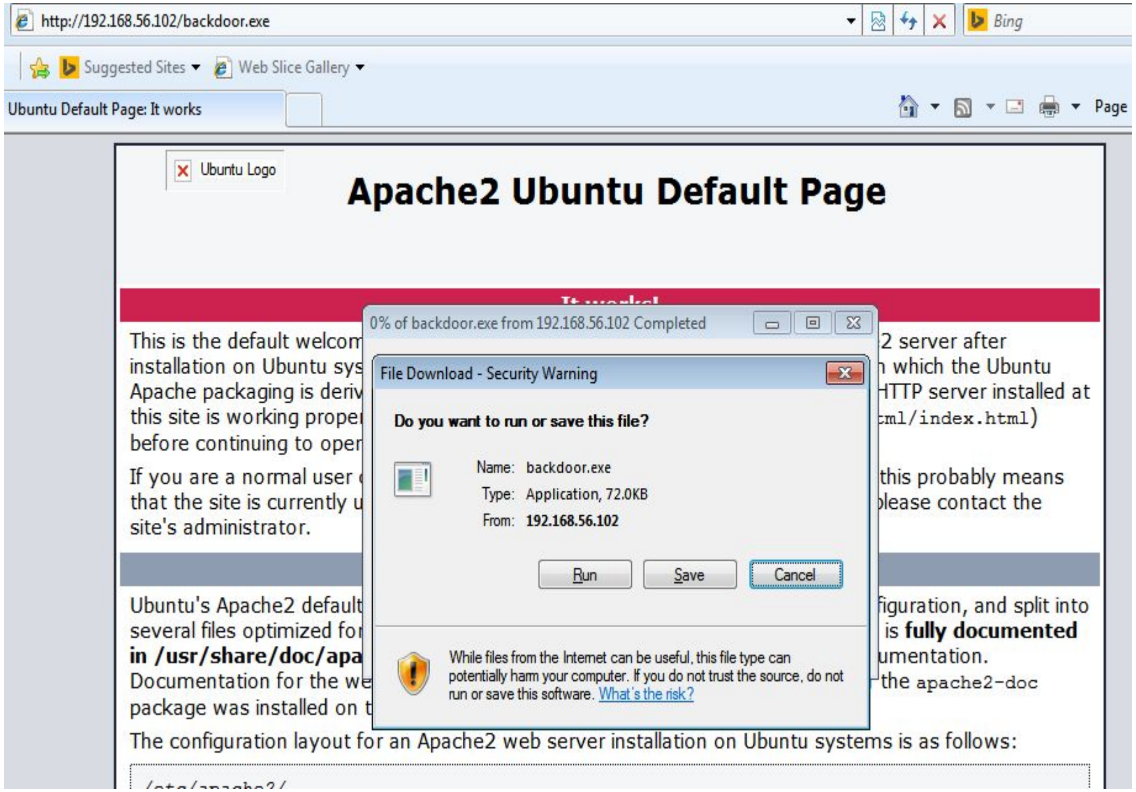
Multi/Handler is used by all Metasploit Payloads in order to establish a connection between the victim and the attacker

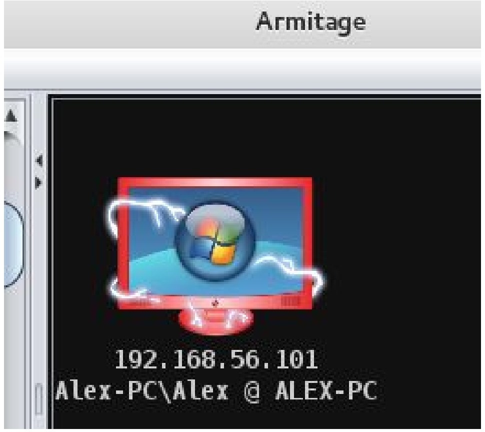
It creates a listener waiting for malware on the victim side to connect



And then?

Once the executable runs and a session is established, Armitage's representation of the victim changes





What are we looking into?

Hobbyists and inexperienced users are more probable to look into tutorials, easy-to-implement attacks that are sure to work

The most common attacks make use of the “**reverse_tcp**” and “**reverse_http(s)**” payloads

They connect back to the attacker and set up a communication according to their title

The presentation will focus on the above payloads

What patterns are we looking for?

Basically... anything that can show any kind of predictability in network behaviour

What patterns are we looking for?

Basically... anything that can show any kind of predictability in network behaviour

What patterns are we looking for?

Basically… anything that can show any kind of predictability in network behaviour

What patterns are we looking for?

Basically… anything that can show any kind of predictability in network behaviour

What did we find?

Transmission of packets every ~60 seconds

5 packets per transmission (652 Bytes per transmission)

reverse_tcp

Randomly chosen port 49163 used in every test

Same packet length, in order, per transmission

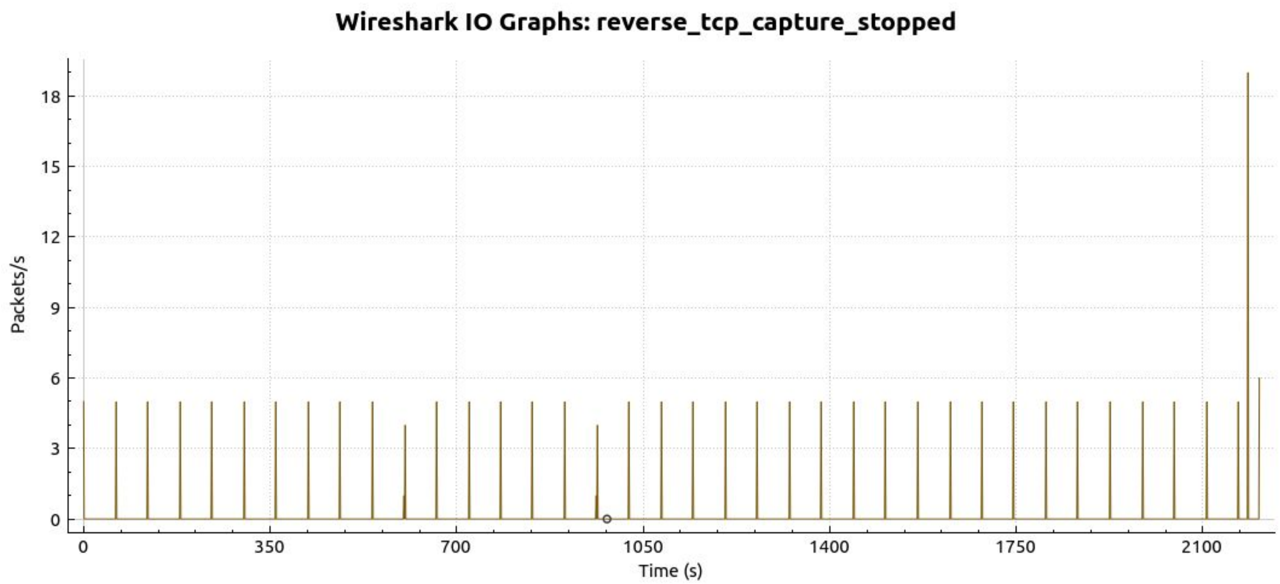
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.56.102	192.168.56.101	TCP	208	4444 → 49163 [PSH, ACK] Seq=1 Ack=1 Win=636 Len=154
4	0.053479	192.168.56.101	192.168.56.102	TCP	128	49163 → 4444 [PSH, ACK] Seq=1 Ack=155 Win=256 Len=74
5	0.053500	192.168.56.102	192.168.56.101	TCP	54	4444 → 49163 [ACK] Seq=155 Ack=75 Win=636 Len=0
6	0.053753	192.168.56.101	192.168.56.102	TCP	208	49163 → 4444 [PSH, ACK] Seq=75 Ack=155 Win=256 Len=154
7	0.053762	192.168.56.102	192.168.56.101	TCP	54	4444 → 49163 [ACK] Seq=155 Ack=229 Win=639 Len=0
46	60.067405	192.168.56.102	192.168.56.101	TCP	208	4444 → 49163 [PSH, ACK] Seq=155 Ack=229 Win=639 Len=154
47	60.115760	192.168.56.101	192.168.56.102	TCP	128	49163 → 4444 [PSH, ACK] Seq=229 Ack=309 Win=255 Len=74
48	60.115797	192.168.56.102	192.168.56.101	TCP	54	4444 → 49163 [ACK] Seq=309 Ack=303 Win=639 Len=0
49	60.116109	192.168.56.101	192.168.56.102	TCP	208	49163 → 4444 [PSH, ACK] Seq=303 Ack=309 Win=255 Len=154
50	60.116120	192.168.56.102	192.168.56.101	TCP	54	4444 → 49163 [ACK] Seq=309 Ack=457 Win=642 Len=0
82	120.359699	192.168.56.102	192.168.56.101	TCP	208	4444 → 49163 [PSH, ACK] Seq=309 Ack=457 Win=642 Len=154
83	120.412497	192.168.56.101	192.168.56.102	TCP	128	49163 → 4444 [PSH, ACK] Seq=457 Ack=463 Win=254 Len=74
84	120.412521	192.168.56.102	192.168.56.101	TCP	54	4444 → 49163 [ACK] Seq=463 Ack=531 Win=642 Len=0
85	120.412710	192.168.56.101	192.168.56.102	TCP	208	49163 → 4444 [PSH, ACK] Seq=531 Ack=463 Win=254 Len=154
86	120.412715	192.168.56.102	192.168.56.101	TCP	54	4444 → 49163 [ACK] Seq=463 Ack=685 Win=644 Len=0

What did we find?

reverse_tcp

When the session closes, the malware exits and has no network presence

The moment the session ends, each test showed a large spike in traffic (10 - 20 packets)



What did we find?

reverse_http(s)

Packet transmission increases from every ~4,5 to 10 seconds

5 packets per transmission (PDU packet size varies per test, 293 - 364)

Randomly chosen port 49164 used in every test

Same packet length, in order, per transmission

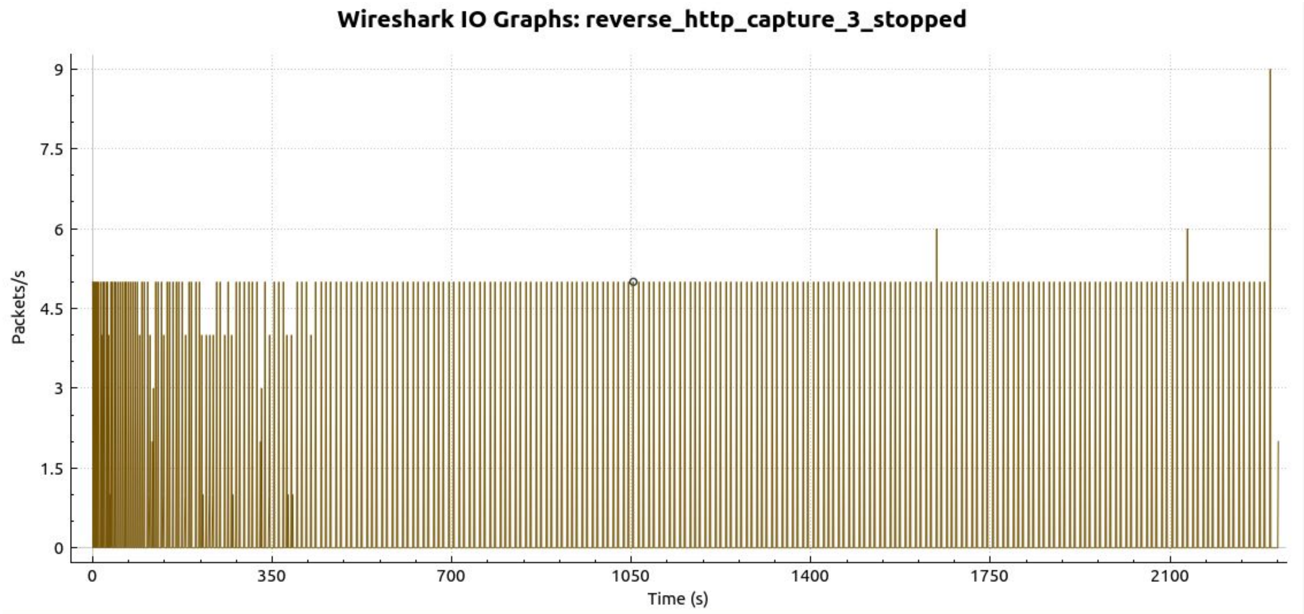
No.	Time	Source	Destination	Protocol	Length	Info
25	3.625116	192.168.56.101	192.168.56.102	TCP	293	[TCP segment of a reassembled PDU]
26	3.625141	192.168.56.101	192.168.56.102	HTTP	60	POST /OlkfiqMWf-QaiRuITCsX5Ajy5Q8EW5P/ HTTP/1.1
27	3.625276	192.168.56.102	192.168.56.101	TCP	54	4444 → 49164 [ACK] Seq=1 Ack=244 Win=182 Len=0
28	3.626047	192.168.56.102	192.168.56.101	HTTP	172	HTTP/1.1 200 OK
29	3.827921	192.168.56.101	192.168.56.102	TCP	60	49164 → 4444 [ACK] Seq=244 Ack=119 Win=251 Len=0
44	8.031380	192.168.56.101	192.168.56.102	TCP	293	[TCP segment of a reassembled PDU]
45	8.031407	192.168.56.101	192.168.56.102	HTTP	60	POST /OlkfiqMWf-QaiRuITCsX5Ajy5Q8EW5P/ HTTP/1.1
46	8.031523	192.168.56.102	192.168.56.101	TCP	54	4444 → 49164 [ACK] Seq=119 Ack=487 Win=182 Len=0
47	8.032314	192.168.56.102	192.168.56.101	HTTP	172	HTTP/1.1 200 OK
48	8.249798	192.168.56.101	192.168.56.102	TCP	60	49164 → 4444 [ACK] Seq=487 Ack=237 Win=256 Len=0
49	12.531739	192.168.56.101	192.168.56.102	TCP	293	[TCP segment of a reassembled PDU]
50	12.531763	192.168.56.101	192.168.56.102	HTTP	60	POST /OlkfiqMWf-QaiRuITCsX5Ajy5Q8EW5P/ HTTP/1.1
51	12.531904	192.168.56.102	192.168.56.101	TCP	54	4444 → 49164 [ACK] Seq=237 Ack=730 Win=182 Len=0
52	12.532579	192.168.56.102	192.168.56.101	HTTP	172	HTTP/1.1 200 OK
53	12.734112	192.168.56.101	192.168.56.102	TCP	60	49164 → 4444 [ACK] Seq=730 Ack=355 Win=256 Len=0

What did we find?

reverse_http(s)

When the session closes, the malware exits and has no network presence

The moment the session ends, each test showed a large spike in traffic (+9 packets)



What about Evasion Techniques?

Antivirus evasion

Encode the generated payload multiple times to increase obfuscation

IDS/IPS evasion

Changing the transport type of the payload, e.g. from TCP to HTTPS

What does it all mean?

There is evidence to suggest the existence of patterns in the network behaviour of certain automatically generated malware

Not all malware behaves the same

Metasploit is an ever changing platform, constantly updating

What is next?

The next step would be to automate this procedure

In a way that false positive occurences would be kept to a minimum

Analyze other frequently used payloads/exploits for multiple platforms

What’s up?

Thank you for your attention. Questions?