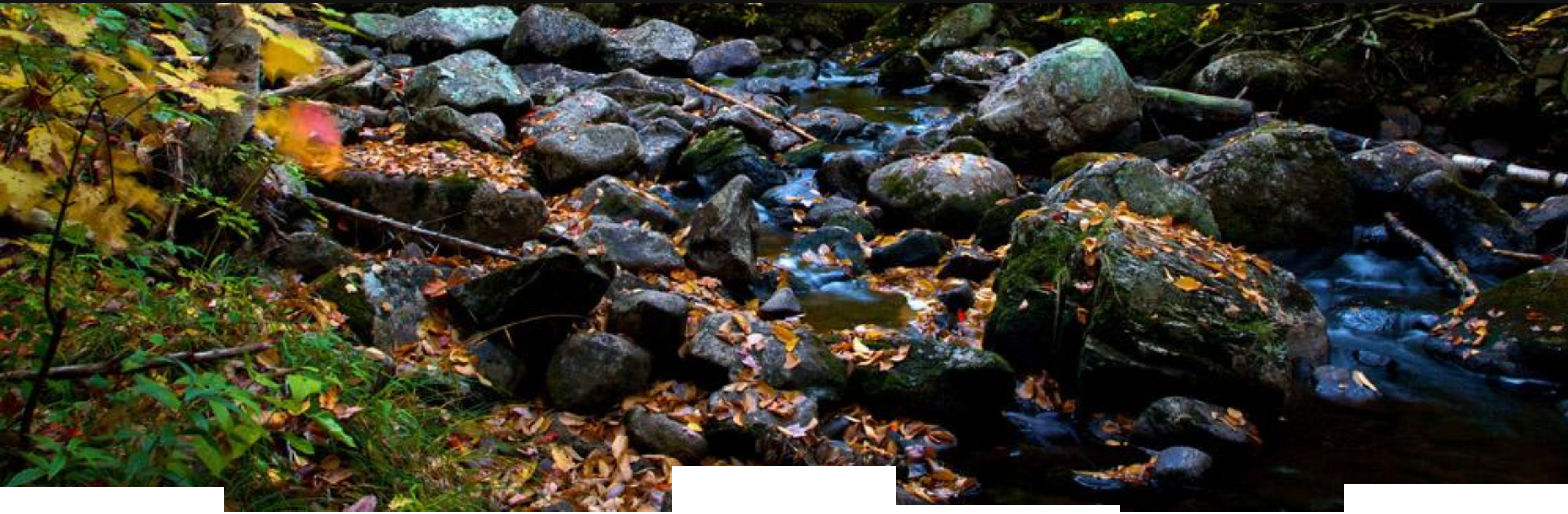
SinFP3

More Than a Complete Framework for Operating System Fingerprinting – v1.0



Patrice <GomoR> Auffret

@PatriceAuffret

@networecon

`whoami`

- Patrice <GomoR> Auffret
  - 10+ years of InfoSec experience
  - www.gomor.org
  - www.protocol-hacking.org (french only)
  - www.secure-side.com (FreeBSD Web hosting company)
  - www.networecon.com (where the tool will be released)
  - Currently working for technicolor (security assessments coordinator)
- Network protocol « Hacker »
  - Net::Frame Perl modules
    - 8021.Q, LLTD, OSPF, IPv4/6, ICMPv4/6, TCP/UDP, STP, …
  - Net::SinFP & Net::SinFP3 Perl modules
    - That is the subject of today
- FreeBSD addict & Perl developer (http://search.cpan.org/~gomor/)

Agenda

- Operating system fingerprinting
  - What is it? (quickly)
  - What is SinFP?
- Limitations of Nmap OS fingerprinting
- SinFP approach to active fingerprinting
- SinFP3 matching algorithm and database
- Demo
- SinFP3 architecture and advances
  - Comparison with previous versions of SinFP
  - Zoom on Input::SynScan, Input::Connect, Input::ArpDiscovery
- SinFP3 passive fingerprinting (if time permits)
- Conclusion

What is operating system fingerprinting (one slide)

- Yes, what's that stuff? *(pretty sure everyone knows already)*
  - The art or remotely identifying the nature of an Operating System by analyzing how its TCP/IP stack is crafting network packets

- Two approaches
  - Active mode
    - Sends probes to elicit responses
    - Analyst decides on the format of requests (very important)
  - Passive mode
    - Listen to the network
    - Analyst does not decide on the format of requests (also very important)

- These two approaches give a different signature (or fingerprint)
  - More on that later (if time permits) …

- Why not simply using application-level « banners »?
  - If you have the choice, use this option
  - Or correlate with OSFP to have a better identification

What is SinFP?

- An Operating System FingerPrinting tool (OSFP)
  - Written in Perl *(the best language, /troll)*
  - Module based, for easy integration in other (Perl?) projects
  - Based on the Net::Frame Perl modules (since SinFP3)
  - 1st tool to implement IPv6 fingerprinting (active and passive) \o/

- History
  - V0.92: June 2005
  - V1.00: March 2006
  - V2.02: September 2006 (complete rewrite)
  - V2.09: March 2011
  - SinFP3 v1.00: now ☺

- Was integrated in BackTrack, but no more in latest versions
  - Who knows why?

Limitations of Nmap OSFP (Nmap 1/2)

- Nmap philosophy: one target IP has only one operating system

- Nmap probes
  - 6 TCP SYN (open port)
  - 1 ICMP echo
  - 1 TCP ECN (open port)
  - 1 TCP null (open port)
  - 1 TCP SYN|FIN|URG|PSH (open port)
  - 1 TCP ACK (open port)
  - 1 TCP SYN (closed port)
  - 1 TCP ACK (closed port)
  - 1 TCP FIN|PSH|URG (closed port)
  - 1 UDP (closed port)

- For a complete fingerprint, target MUST:
  - Have one open TCP port
  - Have one closed TCP port
  - Allow ICMP echo requests
  - Have one closed UDP port (those who answer ICMP port unreachable)

Limitations of Nmap OSFP (Nmap 2/2)

- Problem 1: what if some of target's answers are spoofed?
  - A fitering device in-between answers to:
    - UDP requests
    - Out-of-state probes
  - You have a fingerprint composed of different TCP/IP stacks
    - TurtleOS, anyone?

- Problem 2: filtering, packet normalization and stateful inspection
  - Nmap tests remaining:
    - 6 TCP SYN (open port)
    - 1 TCP ECN (open port) (not sure this one will resist packet normalization)

- Problem 3: easily detected by IDSs/IPSs
  - Too noisy and packet format too easy to classify as Nmap fingerprinting

- Conclusion
  - Nmap is only ok for LAN-side OS fingerprinting in today's Internet conditions

SinFP approach, active mode

- Philisophy: one target IP/port has only one operating system
  - Every probes MUST generate an answer from the true target
  - Every probes MUST reach the true target (filtering evasion)

- We come with 3 TCP packets all targeted at one open TCP port
  - One TCP SYN with just MSS TCP option
    - SinFP2 hadn't options at all, and some TCP/IP stacks don't answer if no option
  - One TCP SYN with many valid TCP options
  - One TCP SYN|ACK (used for LAN-side fingerprinting)

- One operating system has only one signature in the database
  - Matching algorithm takes care of modified fingerprints due to
    - Filtering device in-between (MTU change, for instance)
    - Customization of TCP/IP stack on the system

- During our tests, usually only one TCP SYN is enough to fingerprint reliably a target

A fingerprinting example: Nmap

```
# nmap -P0 -p 80 -O ovh1.secure-side.com
Running (JUST GUESSING): FreeBSD 7.X|6.X|8.X (98%)
Aggressive OS guesses: FreeBSD 7.0-RELEASE (98%), FreeBSD 6.3-RELEASE (98%),
FreeBSD 7.1-PRERELEASE 7.2-STABLE (98%), FreeBSD 7.2-RELEASE - 8.0-RELEASE
(94%), FreeBSD 8.1-RELEASE (94%), FreeBSD 7.1-PRERELEASE - 7.3-RELEASE (93%),
FreeBSD 7.1-RELEASE - 9.0-CURRENT (93%), FreeBSD 8.0-STABLE (93%), FreeBSD
7.0-STABLE (93%), FreeBSD 7.0-RELEASE - 8.0-STABLE (92%)
```

A fingerprinting example: SinFP3

```
# sinfp3.pl -input-ipport -target ovh1.secure-side.com -port 80 -threshold 70 –active-2
Result for target [213.251.166.100]:80:
S1: B11113 F0x12 W65535 O0204ffff M1460 S0 L4
S2: B11113 F0x12 W65535 O0204ffff010303ff0402080affffffff44454144 M1460 S3 L20
IPv4: [score:100]: BH0FH0WH0OH0MH0SH0LH0/S1S2: BSD: OSS: FreeBSD: 7.4 (7.4-RELEASE)
IPv4: [score:100]: BH0FH0WH0OH0MH0SH0LH0/S1S2: BSD: OSS: FreeBSD: 7.0 (7.0-RELEASE)
IPv4: [score:100]: BH0FH0WH0OH0MH0SH0LH0/S1S2: BSD: OSS: FreeBSD: 7.3 (7.3-RELEASE)
IPv4: [score:100]: BH0FH0WH0OH0MH0SH0LH0/S1S2: BSD: OSS: FreeBSD: 8.1 (8.1-RELEASE)
IPv4: [score:100]: BH0FH0WH0OH0MH0SH0LH0/S1S2: BSD: OSS: FreeBSD: 8.0 (8.0-RELEASE)
IPv4: [score:100]: BH0FH0WH0OH0MH0SH0LH0/S1S2: BSD: OSS: FreeBSD: 7.1 (7.1-RELEASE)
IPv4: [score:100]: BH0FH0WH0OH0MH0SH0LH0/S1S2: BSD: OSS: FreeBSD: 8.2 (8.2-RELEASE)
IPv4: [score:100]: BH0FH0WH0OH0MH0SH0LH0/S1S2: BSD: OSS: FreeBSD: 8.3 (8.3-RELEASE)
IPv4: [score:100]: BH0FH0WH0OH0MH0SH0LH0/S1S2: BSD: OSS: FreeBSD: 7.2 (7.2-RELEASE)
IPv4: [score:94]: BH0FH0WH0OH0MH0SH1LH0/S1S2: BSD: OSS: FreeBSD: 9.0 (9.0-RELEASE)
```

SinFP3 matching algorithm (signatures 1/8)

- Binary flags, comparison between probe and response IP/TCP headers

S1: **B11113** F0x12 W65535 O0204ffff M1460 S0 L4

S2: **B11113** F0x12 W65535 O0204ffff010303ff0402080affffffff44454144 M1460 S3 L20

S3: **B11120** F0x04 W0 O0 M0 S0 L0

- Some comparison methods were taken from Nmap (O2)
  - Comparison between TCP probes and replies on SEQ and ACK numbers
  - Not anymore binary, but kept the name

SinFP3 matching algorithm (signatures 2/8)

- TCP flags

S1: B11113 **F0x12** W65535 O0204ffff M1460 S0 L4

S2: B11113 **F0x12** W65535 O0204ffff010303ff0402080affffffff44454144 M1460 S3 L20

S3: B11120 **F0x04** W0 O0 M0 S0 L0

- Maybe a target will answer with more flags than SYN|ACK or RST?
  - Never seen yet

SinFP3 matching algorithm (signatures 3/8)

- TCP window size

S1: B11113 F0x12 **W65535** O0204ffff M1460 S0 L4

S2: B11113 F0x12 **W65535** O0204ffff010303ff0402080affffffff44454144 M1460 S3 L20

S3: B11120 F0x04 **W0** O0 M0 S0 L0

- One of the most important element

SinFP3 matching algorithm (signatures 4/8)

- TCP options, values are extracted (like MSS, WScale)

S1: B11113 F0x12 W65535 **O0204ffff** M1460 S0 L4

S2: B11113 F0x12 W65535 **O0204ffff010303ff0402080affffffff44454144** M1460 S3 L20

S3: B11120 F0x04 W0 **O0** M0 S0 L0

- The most important element
  - Number and order of TCP options is the best differientor between OSs

- Data may be returned from the target
  - It is integrated into this element
  - HP-UX loves to add « No TCP » data like this:

S3: B11120 F0x04 W0 **O4e6f20544350** M0 S0 L6

SinFP3 matching algorithm (signatures 5/8)

- Extracted MSS value

S1: B11113 F0x12 W65535 O0204ffff **M1460** S0 L4

S2: B11113 F0x12 W65535 O0204ffff010303ff0402080affffffff44454144 **M1460** S3 L20

S3: B11120 F0x04 W0 O0 **M0** S0 L0

- By extracting it, we make it easier to write our deformation masks
  - Explanation will come

SinFP3 matching algorithm (signatures 6/8)

- Extracted WScale value

S1: B11113 F0x12 W65535 O0204ffff M1460 **S0** L4

S2: B11113 F0x12 W65535 O0204ffff010303ff0402080affffffff44454144 M1460 **S3** L20

S3: B11120 F0x04 W0 O0 M0 **S0** L0

- Same here, easy to write deformation masks

SinFP3 matching algorithm (signatures 7/8)

- Length of TCP options (in bytes)

S1: B11113 F0x12 W65535 O0204ffff M1460 S0 **L4**

S2: B11113 F0x12 W65535 O0204ffff010303ff0402080affffffff44454144 M1460 S3 **L20**

S3: B11120 F0x04 W0 O0 M0 S0 **L0**

SinFP3 matching algorithm (signatures 8/8)

- Complete IPv4 active signature (FreeBSD 8.3-RELEASE)

S1: B11**1**13 F0x12 W65535 O0204ffff M**1460** S0 L4

S2: B11**1**13 F0x12 W65535 O0204ffff010303ff0402080affffffff44454144 M**1460** S3 L20

S3: B11**1**20 F0x04 W0 O0 M0 S0 L0

- Complete IPv6 active signature (FreeBSD 8.3-RELEASE)

S1: B11**0**13 F0x12 W65535 O0204ffff M**1440** S0 L4

S2: B11**0**13 F0x12 W65535 O0204ffff010303ff0402080affffffff44454144 M**1440** S3 L20

S3: B10**0**20 F0x04 W0 O0 M0 S0 L0

- Complete IPv4 passive signature (Windows 7)

SP: F0x02 W8192 O0204ffff010303ff01010402 M**1460** S8 L12

- Complete IPv6 passive signature (Windows 7)

SP: F0x02 W8192 O0204ffff010303ff01010402 M**1420** S8 L12

SinFP3 matching algorithm (masks 1/4)

- 3 level of deformation
  - Heuristic0: no deformation
  - Heuristic1: minor deformations
  - Heuristic2: major deformations

- Deformation mask takes care of devices modifying packets
  - No need to add many signatures for one same operating system
  - So, number of signatures is far less than from Nmap’s database

- Example: all elements with heuristic1 deformation:

S1H1: B...13 F0x12 W6[45]... O0204ffff M1[34].. S. L4

S2H1: B...13 F0x12 W6[45]... O0204ffff(?:01)?(?:0303ff)?(?:0402)?(?:080affffffff44454144)? M1[34].. S. L(?:8|9|[12].)

S3H1: B...20 F0x04 W0 O0 M0 S. L0

SinFP3 matching algorithm (masks 2/4)

- Non-deformed signature
  - Match score: 100% (BH0FH0WH0OH0**MH0**SH0LH0)

S1: B11113 F0x12 W65535 O0204ffff **M1460** S0 L4

S2: B11113 F0x12 W65535 O0204ffff010303ff0402080affffffff44454144 **M1460** S3 L20

S3: B11120 F0x04 W0 O0 M0 S0 L0

SinFP3 matching algorithm (masks 3/4)

- Deformed signature because of reduced MTU (classic stuff)
  - Match score: 98% (BH0FH0WH0OH0**MH1**SH0LH0)

S1: B11113 F0x12 W65535 O0204ffff **M1452** S0 L4

S2: B11113 F0x12 W65535 O0204ffff010303ff0402080affffffff44454144 **M1452** S3 L20

S3: B11120 F0x04 W0 O0 M0 S0 L0

SinFP3 matching algorithm (masks 4/4)

- Deformed signature because of reduced MTU (classic stuff)
  - Match score: 98% (BH0FH0WH0OH0**MH1**SH0LH0)

S1: B11113 F0x12 W65535 O0204ffff **M1[34]..** S0 L4

S2: B11113 F0x12 W65535 O0204ffff010303ff0402080affffffff44454144 **M1[34]..** S3 L20

S3: B11120 F0x04 W0 O0 M0 S0 L0

- Each element (B, F, W, O, M, S, L) has a weight
  - No deformation means higher weight (BH0, FH0, WH0, …)
  - Most discriminent elements have higher weights (window size, options)
  - Match score is calculated by additioning these match scores

SinFP3 matching algorithm (intersection)

- Every element has heurisitic0 (no deformation), heuristic1 and heuristic2 patterns in the database

- A match is found when:
  - Intersection exists between S1, S2 and S3 signatures
  - And by applying deformation masks when no match is found
  - Highest score are kept as a matched fingerprint
  - Then S1 intersection with S2, then only S2

- For IPv6:
  - A matching signature is found: OK
  - Nothing found, try searching against IPv4 signatures
    - This works great, thanks to deformation masks

- For passive fingerprinting:
  - Same algorithm, but against passive signatures

SinFP3 database

- SQLite based
  - Table Signature (active ones; 275 at this day)
  - Table SignatureP (passive ones; 21 at this day)

- Not every signature is integrated
  - Only taken from best conditions (usually target is installed on a VM)
  - Only one signature per operating system version
  - Trusted and untrusted signatures (flag in the database)

- All pcap traces are kept
  - Ready for changes on analysis in the future
  - A pretty good pcap database of operating systems
  - Complete SinFP exchange for active mode, and SYN only for passive mode

- Need contributors for passive signature
  - => sinfp[at]networecon.com

Demo

- ARP discovery, IPv4 active fingerprinting
  - For IPv6 mode, it is as easy as adding -6 option

- Default modules
  - Input::SynScan (-input-synscan)
  - DB::SinFP3 (-db-sinfp3)
  - Mode::Active (-mode-active)
  - Search::Active (-search-active)
  - Output::Console (-output-console)

- Command lines

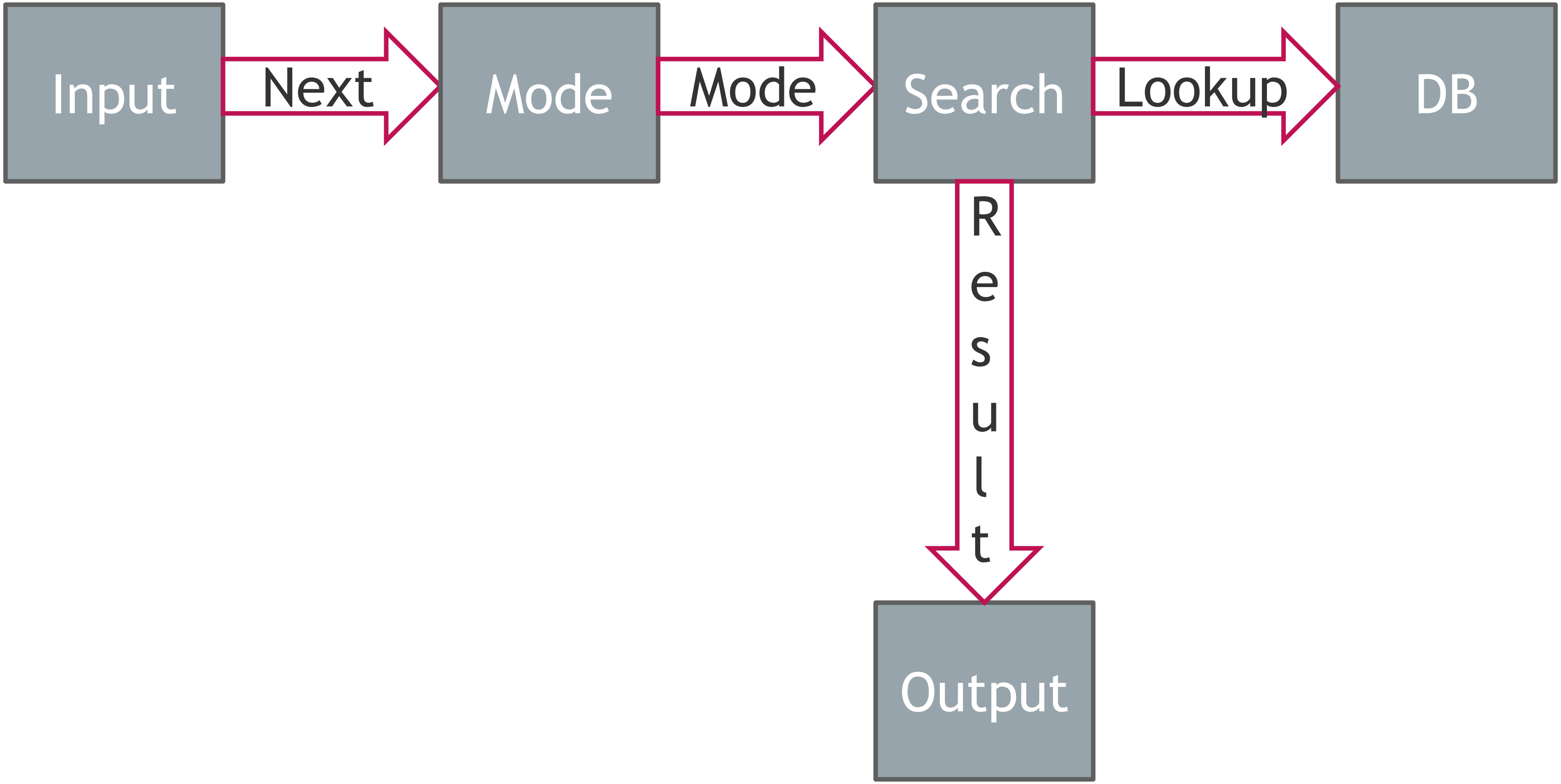
```
# sinfp3.pl -input-arpdiscover -output-pcap
% sinfp3.pl -input-pcap -pcap-file '*.pcap' -output-csv –threshold 80
% sinfp3.pl -db-null -search-null -mode-null -input-null -output-ubigraph
```

SinFP3 architecture and advances (1/2)

- Architecture and features
  - Plugin-based
  - Input, Mode, Search, DB, Output plugins

- Improvements on Active and Passive modes
  - Matching algorithm
    - Deformation masks were written manually
    - No match score
  - Probe requests
    - Probe P1 has now a TCP MSS option
  - Autonomous passive mode
    - Passive signature database is no more correlated with active one
  - And of course, the plugin-based architecture
    - Allowing massive parallel scanning (for instance)

SinFP3 architecture and advances (2/2)



Currently implemented plugins

- Input modules
  - Input::Pcap, Input::IpPort, Input::SynScan, Input::ArpDiscover, Input::Sniff
  - Input::Signature, Input::SignatureP, Input::Connect
- DB modules
  - DB::SinFP3
- Mode modules
  - Mode::Active, Mode::Passive
- Search modules
  - Search::Active, Search::Passive
- Output modules
  - Output::Console, Output::Pcap, Output::CSV, Output::OsOnly, Output::OsVersionFamily, Output::Ubigraph

Zoom on Input::SynScan

- Written in Perl/XS/C
  - IPv4 and IPv6 ready
  - Efficient enough
  - Deterministic
  - 20 minutes for TOP10 ports against a C-class
    - Default: 200 packets per second, 3 tries (around 10 kB/s)
  - KISS algorithm (do it yourself ;))

- Writes TCP packets directly at layer 4
  - Don’t bother with computing checksums and other IP headers
  - Works under GNU/Linux and BSD systems
  - Uses SinFP3 magic SYN packet

- Scan once, replay fingerprinting
  - Output::Pcap, then Input::Pcap

Zoom on Input::Connect

- Because SYN|ACK fingerprinting was a failure …

- Use TCP connect() and send a classic « GET / HTTP/1.0 »
  - A listener is catching SYN probe and SYN|ACK reply
  - Mode::Active generates the fingerprint
  - Search::Active searches a matching signatures

- Works great from Linux (only?)
  - Cause the SYN probe is the same used in SinFP active mode
  - Same window size and TCP options

- Nearly stealthiest option for fingerprinting
  - Not seen as active fingerprinting by a potential target IDS/IPS

Zoom on Input::ArpDiscover

- On your LAN (of course)
  - Performs a standard ARP scanning against all LAN IP addresses
  - Gathers all live hosts
  - Then performs an active fingerprinting of all live hosts
    - Currently, you have to specify which target ports to test

- For IPv6
  - Performs a standard ARP scanning against all LAN IPv4 addresses
  - Gathers all live hosts
  - Apply EUI-64 transform against MAC addresses
    - You have the list of auto-configured link-local IPv6 addresses
  - Then performs an active fingerprinting of all live hosts

- For IPv6, you didn’t thought of scanning the fe80::/64, did you?

SinFP passive fingerprinting (1/2) (time?)

- p0fv3
  - IPv4 and IPv6 passive fingerprinting
  - TCP SYN and TCP SYN|ACK
  - A very comprehensive signature database

- SinFP2
  - IPv4 and IPv6 passive fingerprinting
  - TCP SYN and TCP SYN|ACK
  - No passive signature in the database
  - A transform was applied on a fingerprint to make use of active signatures
    - It was failure *

- Conclusion: SYN|ACK fingerprinting does not work
  - SYN|ACKs are generated compared to the original SYN probe
  - You don't control how SYNs are generated by different equipments you are monitoring
  - So, there exists a multitude of SYN|ACK fingerprints for one unique operating system (p0fv3 uses this approach)

* @GoulagParkinson: thanks for catching this up

SinFP passive fingerprinting (2/2) (time?)

- SinFP3 approach:
  - Only TCP SYNs are fingerprinted
  - Signature database schema update to have passive signatures appart from active signatures

- But still work in progress, not many signatures right now
  - Need contributions, please send signatures to sinfp[AT]networecon.com
    - I may have said it already ;)

```
% sqlite3 bin/sinfp3.db
sqlite> select count(*) from SignatureP;
21
sqlite> select count(*) from Signature;
275
```

Conclusion

- Improvements on matching algorithm
  - No more manual deformation masks
  - Computes a matching score for easy human comprehension

- Improvements on architecture allowing to
  - Write new modules, like new matching algorithms or output methods
  - Perform more than OS fingerprinting

- Improvements on passive fingerprinting
  - But needs more signature (did I said that already?)

- Many more features
  - Plugin to add signatures to the database by yourself
  - Update database with –update-db
  - Logging modules
  - Design your own plugins … limitless?

- Follow @networecon to get informed of releases
  - http://www.networecon.com/

Follow me @PatriceAuffret @networecon

Questions? (I can haz a beer now?)

http://www.networecon.com/

This document is for background informational purposes only. Some points may, for example, be simplified. No guarantees, implied or otherwise, are intended