The menace came from below

Éric Leblond, Victor Julien

OISF

Hack.lu 2012

Eric Leblond a.k.a Regit

- French
- Network security expert
- Free Software enthousiast
- NuFW project creator (Now ufwi), EdenWall co-founder
- Netfilter developer:
  - Ulogd2: Netfilter logging daemon
  - Misc contributions:
    - NFQUEUE library and associates
    - Source NAT randomisation (defeat Kaminsky's DNS attack)
- Currently:
  - Independant security consultant
  - Suricata IDS/IPS funded developer

Victor Julien a.k.a Inliniac

- Dutch
- Open Source Developer and Contractor
- Vuurmuur Firewall project creator
- Suricata IDS/IPS lead developer

What is Suricata

- IDS and IPS engine
- Get it here:
  `http://www.suricata-ids.org`
- Open Source (GPLv2)
- Funded by US government and consortium members
- Run by Open Information Security Foundation (OISF)
- More information about OISF at
  `http://www.openinfosecfoundation.org/`

Suricata Features

- High performance, scalable through multi threading
- Protocol identification
- File identification, extraction, on the fly MD5 calculation
- TLS handshake analysis, detect/prevent things like Diginotar
- Hardware acceleration support:
  - Endace
  - Napatech,
  - CUDA
  - PF_RING

Suricata Features

- Rules and outputs compatible to Snort syntax
- useful logging like HTTP request log, TLS certificate log
- (experimental) Lua scripting for detection

1. Introduction
    - Netfilter and the Conntrack
    - Degree of freedom in Netfilter helpers
2. Multi-layer attack
    - Conditions and principles
    - FTP case
    - Checkpoint
    - Others protocols
3. Impact and existing protection
    - Netfilter
    - Detecting the attack
4. Protocol analysis attack
    - Protocol analysis
    - Low TTL attack
    - The attack on nDPI and Suricata
5. Conclusion

Netfilter

Definition

Packet filtering framework inside the Linux 2.4.x to 3.x kernel series.

Netfilter

Definition

Packet filtering framework inside the Linux 2.4.x to 3.x kernel series.

Features

- Stateful and stateless packet filtering (IPv4 and IPv6).
- Network address and port translation (NAT).
- Multiple layers of API's for 3rd party extensions.

Netfilter

Definition

Packet filtering framework inside the Linux 2.4.x to 3.x kernel series.

Features

- Stateful and stateless packet filtering (IPv4 and IPv6).
- Network address and port translation (NAT).
- Multiple layers of API's for 3rd party extensions.

Iptables

- Command line utility to do operation on rules.
- It has access to all Netfilter features.
- Two utilities: iptables for IPv4, ip6tables for IPv6.

```
iptables –A FORWARD –p tcp —syn —dport 80 \
         –m connlimit —connlimit–above 2 –j REJECT
```

Application Level Gateway

Non-linear protocol

One can find protocols such as FTP or SIP:

- They rely on a signalling channel.
- It is used to setup dynamic connections.

Application Level Gateway

Non-linear protocol

One can find protocols such as FTP or SIP:

- They rely on a signalling channel.
- It is used to setup dynamic connections.

Application Level Gateway (ALG)

- ALGs search the traffic for command messages.
- They extract information on the expected connections.
- Each expectation:
  - includes information on a potential connection.
  - is associated to a timeout.
- New connection matching an expectation can be accepted.

The example of FTP

FTP client

```
Logged in to ftp.lip6.fr.
ncftp / > ls
etc/          jussieu/    lip6/
```

Tcpdump

```
195.83.118.1.21 > 10.62.101.203.52994
195.83.118.1.21 > 10.62.101.203.52994
10.62.101.203.57636 > 195.83.118.1.51155
10.62.101.203.52994 > 195.83.118.1.21
195.83.118.1.51155 > 10.62.101.203.57636
```

The example of FTP

FTP client

```
Logged in to ftp.lip6.fr.
ncftp / > ls
etc/          jussieu/    lip6/
```

Tcpdump

```
195.83.118.1.21 > 10.62.101.203.52994
195.83.118.1.21 > 10.62.101.203.52994
10.62.101.203.57636 > 195.83.118.1.51155
10.62.101.203.52994 > 195.83.118.1.21
195.83.118.1.51155 > 10.62.101.203.57636
```

Protocol

```
C: PASV
S: 227 Entering Passive Mode (195,83,118,1,199,211)
C: MLSD
S: 150 Opening ASCII mode data connection for 'MLSD'.
S: 226 MLSD complete.
C: QUIT
```

The example of FTP

FTP client

```
Logged in to ftp.lip6.fr.
ncftp / > ls
etc/          jussieu/    lip6/
```

Tcpdump

```
195.83.118.1.21 > 10.62.101.203.52994
195.83.118.1.21 > 10.62.101.203.52994
10.62.101.203.57636 > 195.83.118.1.51155
10.62.101.203.52994 > 195.83.118.1.21
195.83.118.1.51155 > 10.62.101.203.57636
```

Protocol

```
C: PASV
S: 227 Entering Passive Mode (195,83,118,1,199,211)
C: MLSD
S: 150 Opening ASCII mode data connection for 'MLSD'.
S: 226 MLSD complete.
C: QUIT
```

Netfilter

```
# conntrack -E expect
    [NEW] 300 proto=6 src=10.62.101.203 dst=195.83.118.1 sport=0 dport=51155
[DESTROY] 300 proto=6 src=10.62.101.203 dst=195.83.118.1 sport=0 dport=51155
```

Details of Netfilter implementation

ALGs in Netfilter

- ALGs are called *Helpers*.
- Each protocol is implemented as a kernel module.
- Loading options can be used to configure the helper.
- Fine-grained setup can be achieved with the CT iptables target.

Details of Netfilter implementation

ALGs in Netfilter

- ALGs are called *Helpers*.
- Each protocol is implemented as a kernel module.
- Loading options can be used to configure the helper.
- Fine-grained setup can be achieved with the CT iptables target.

Current modules list in Vanilla linux kernel

amanda
ftp
proto_udplite
netbios_ns
pptp
proto_gre
sane
snmp
broadcast
h323
irc
tftp
proto_dccp
proto_sctp
sip

Do I use helpers?

- What happens if I load a helper?

Do I use helpers?

- What happens if I load a helper?
- Can a user send crafted messages and go freely through the firewall?

Do I use helpers?

- What happens if I load a helper?
- Can a user send crafted messages and go freely through the firewall?
- Do helpers transform my firewall in openbar?

Do I use helpers?

- What happens if I load a helper?
- Can a user send crafted messages and go freely through the firewall?
- Do helpers transform my firewall in openbar?

Do I use helpers?

- What happens if I load a helper?
- Can a user send crafted messages and go freely through the firewall?
- Do helpers transform my firewall in openbar?

- A study is needed.
- Let's look at the helpers.

Global analysis

Sane defaults

- Dangerous extensions of protocols have been disabled.
- If we study the attack of client on a server:
  - It is impossible to open arbitrary connections to the server.
  - The level of security is acceptable.

Global analysis

Sane defaults

- Dangerous extensions of protocols have been disabled.
- If we study the attack of client on a server:
  - It is impossible to open arbitrary connections to the server.
  - The level of security is acceptable.

In the limit of protocols

- Security is ensured with regard to the protocol usability.
- IRC helper is really user-friendly.

FTP analysis

If we follow RFC (*loose* $= 0$).

- A FTP server can participate to the initialization of a connection from client to another server.
- It can open arbitrary connections through the firewall.

FTP analysis

If we follow RFC (*loose* $= 0$).

- A FTP server can participate to the initialization of a connection from client to another server.
- It can open arbitrary connections through the firewall.

If we care about security (*loose* $= 1$).

- Expectation are statically bound to the server address.
- The possible openings are acceptable.
- This is the default value.

IRC analysis

The DCC command

DCC command enables transfer between end-point.

- It is impossible to know the source address.
- Destination port is fixed by the client.

IRC analysis

The DCC command

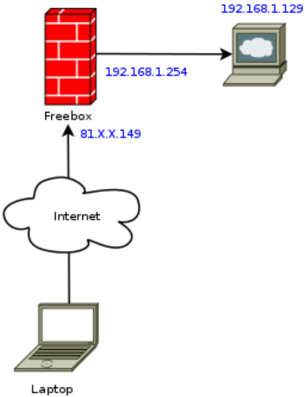
DCC command enables transfer between end-point.

- It is impossible to know the source address.
- Destination port is fixed by the client.

Consequences

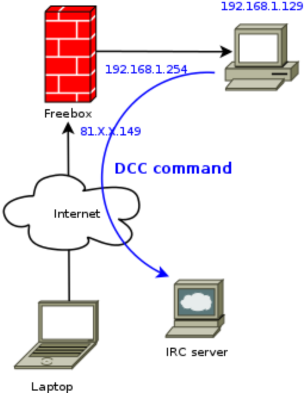
- Allowing DCC is thus allowing client to enable arbitrary connection to his IP.
- Client computer is given a complete freedom of connection opening.

Using DCC command



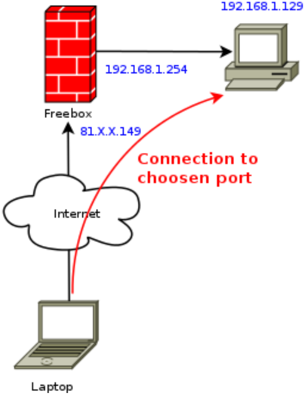
- Client NATed behind firewall, port *N* is closed

Using DCC command



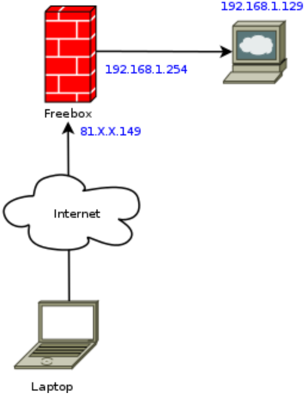
- Client NATed behind firewall, port *N* is closed
- Client sends a DCC command to a valid IRC server

Using DCC command



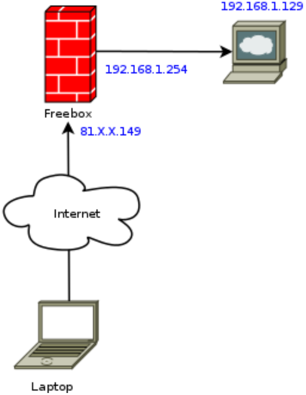
- Client NATed behind firewall, port *N* is closed
- Client sends a DCC command to a valid IRC server
- Firewall creates expectation and laptop can open a connection

Demonstration of DCC usage



Video

Demonstration of DCC usage



Video

Let’s connect from Internet to port 6000 of a NATed client.

1. Introduction
   - Netfilter and the Conntrack
   - Degree of freedom in Netfilter helpers
2. Multi-layer attack
   - Conditions and principles
   - FTP case
   - Checkpoint
   - Others protocols
3. Impact and existing protection
   - Netfilter
   - Detecting the attack
4. Protocol analysis attack
   - Protocol analysis
   - Low TTL attack
   - The attack on nDPI and Suricata
5. Conclusion

Objective

Determine if it is possible *as client* to trigger unwanted behaviour

- Can we open arbitrary pinholes through a firewall?
- Can we open more ports on a server?
- Can we access to badly protected service ?
  - Such as an internal database
  - Such as vulnerable services

Objective

Determine if it is possible *as client* to trigger unwanted behaviour

- Can we open arbitrary pinholes through a firewall?
- Can we open more ports on a server?
- Can we access to badly protected service ?
  - Such as an internal database
  - Such as vulnerable services

Study of helpers has shown that it is not possible out of the box

- Client capabilities are always limited.
- Dangerous extensions have been blocked.
- An alternative approach should be found.

Attack description

Existing attacks force server to send command

- By forcing it to send error message containing a command
- By jumping between helpers

Attack description

Existing attacks force server to send command

- By forcing it to send error message containing a command
- By jumping between helpers

"On est jamais mieux servi que par soi-même"

- Attacker can simply send crafted packets for the server
- If he is on on ethernet network connected to the server
  - Packet is seen at Ethernet level as coming from client
  - Packet at IP level is coming from server and containing a command

Attack description

Existing attacks force server to send command

- By forcing it to send error message containing a command
- By jumping between helpers

"On est jamais mieux servi que par soi-même"

- Attacker can simply send crafted packets for the server
- If he is on on ethernet network connected to the server
  - Packet is seen at Ethernet level as coming from client
  - Packet at IP level is coming from server and containing a command

Man on the side

- Attacker is part of the conversation
- This is not TCP session hijacking
- Packet parameters are build using standard algorithms

Attack description for FTP

1. The attacker sniffs traffic coming from a FTP server.

Attack description for FTP

1. The attacker sniffs traffic coming from a FTP server.
2. He opens a connection to the FTP server.

Attack description for FTP

1. The attacker sniffs traffic coming from a FTP server.
2. He opens a connection to the FTP server.
3. He forges a packet using the last packet received from server:
   - Invert source and destination ethernet address.
   - Increment IP ID. Set TCP sequence number correctly using traffic data.
   - Modify payload to a server command choosing parameters.

     ```
     227 Entering Passive Mode (192,168,2,2,12,234)
     ```
   - Update all checksums and lengths.

Attack description for FTP

1. The attacker sniffs traffic coming from a FTP server.
2. He opens a connection to the FTP server.
3. He forges a packet using the last packet received from server:
   - Invert source and destination ethernet address.
   - Increment IP ID. Set TCP sequence number correctly using traffic data.
   - Modify payload to a server command choosing parameters.

     ```
     227 Entering Passive Mode (192,168,2,2,12,234)
     ```
   - Update all checksums and lengths.
4. The attacker sends the forged packet to the firewall.

Attack description for FTP

1. The attacker sniffs traffic coming from a FTP server.
2. He opens a connection to the FTP server.
3. He forges a packet using the last packet received from server:
    - Invert source and destination ethernet address.
    - Increment IP ID. Set TCP sequence number correctly using traffic data.
    - Modify payload to a server command choosing parameters.

      ```
      227 Entering Passive Mode (192,168,2,2,12,234)
      ```

    - Update all checksums and lengths.
4. The attacker sends the forged packet to the firewall.
5. The firewall creates an expectation for a connection to 192.168.2.2 on port 3306.

Attack description for FTP

1. The attacker sniffs traffic coming from a FTP server.
2. He opens a connection to the FTP server.
3. He forges a packet using the last packet received from server:
   - Invert source and destination ethernet address.
   - Increment IP ID. Set TCP sequence number correctly using traffic data.
   - Modify payload to a server command choosing parameters.

     ```
     227 Entering Passive Mode (192,168,2,2,12,234)
     ```
   - Update all checksums and lengths.
4. The attacker sends the forged packet to the firewall.
5. The firewall creates an expectation for a connection to 192.168.2.2 on port 3306.
6. The attacker connects to 192.168.2.2 on port 3306.

Please ouvrir: opensvp

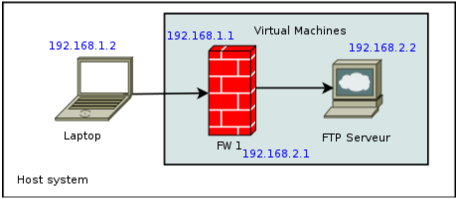
A tool to implement firewall attack

- Implement all attacks described in this talk
- Published under GPLv3 licence
- Available at `https://home.regit.org/software/opensvp/`

written in Python

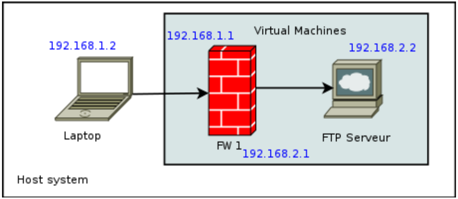
- scapy is used for packet manipulation
- Get scapy and its doc at: `http://www.secdev.org/projects/scapy/`
- the rest is plain Python

Demonstration on Netfilter



Video

Demonstration on Netfilter



Video

Let's have firewall with a filtering policy allowing only port 21 and open a connection to port 22 on a FTP server.

Policy violation

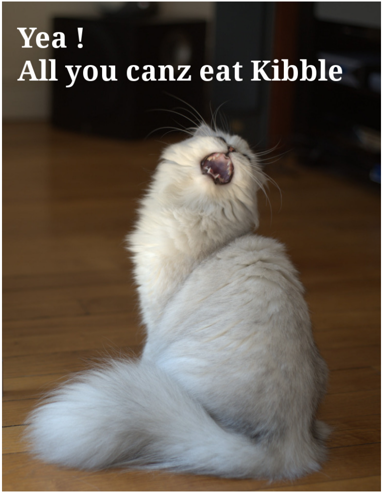
- We’ve manage to open a connection to port 22

Policy violation

- We've manage to open a connection to port 22
- With a filtering policy that does not allow it.

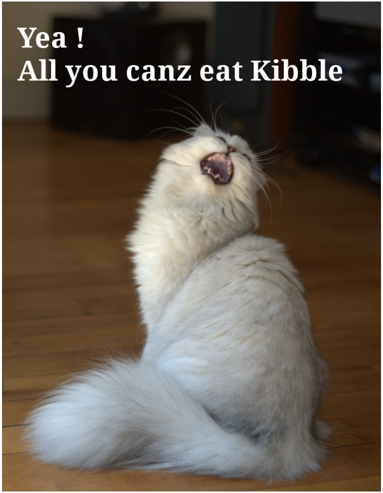
Policy violation

- We've manage to open a connection to port 22
- With a filtering policy that does not allow it.



Policy violation

- We've manage to open a connection to port 22
- With a filtering policy that does not allow it.
- Easy little cat, easy!



Counter-measures

- Anti-spoofing is sufficient to block the attack.
- Reverse path filtering is our friend:
    - Only accept packet coming to an interface if we have a route to the source IP.
    - This will avoid that the kernel handles the attack packet.
- Is this that easy to be protected?

Counter-measures

- Anti-spoofing is sufficient to block the attack.
- Reverse path filtering is our friend:
  - Only accept packet coming to an interface if we have a route to the source IP.
  - This will avoid that the kernel handles the attack packet.
- Is this that easy to be protected? **Yes**

Counter-measures

- Anti-spoofing is sufficient to block the attack.
- Reverse path filtering is our friend:
  - Only accept packet coming to an interface if we have a route to the source IP.
  - This will avoid that the kernel handles the attack packet.
- Is this that easy to be protected? Yes
- But wait, there is still some surprise.

Checkpoint setup

Checkpoint absolute newbie

- I did not read the documentation.
- Why should I? I’m working on firewalls for many years.

Checkpoint setup

Checkpoint absolute newbie

- I did not read the documentation.
- Why should I? I’m working on firewalls for many years.

Used software

- Demonstration version.
- Minimal features installed.

Checkpoint setup

Checkpoint absolute newbie

- I did not read the documentation.
- Why should I? I’m working on firewalls for many years.

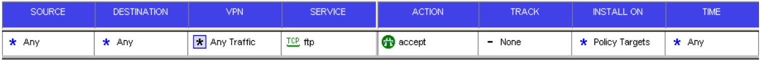
Used software

- Demonstration version.
- Minimal features installed.
- Per default installation.

Demonstration setup

Demonstration setup

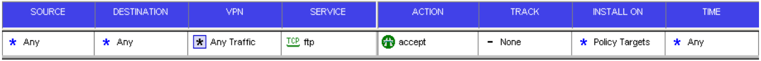
- Let’s do a filtering policy with a single FTP allowed rule ;



SOURCE	DESTINATION	VPN	SERVICE	ACTION	TRACK	INSTALL ON	TIME
Any	Any	Any Traffic	TCP ftp	accept	None	Policy Targets	Any

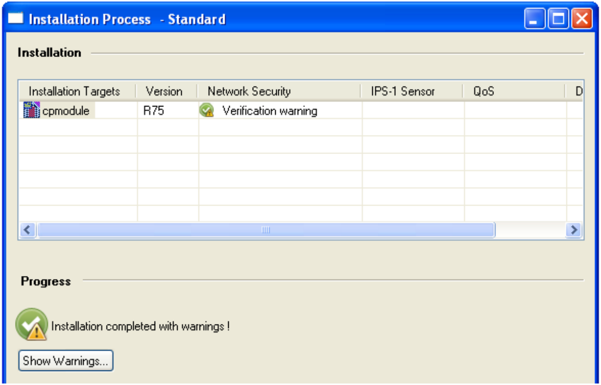
Demonstration setup

- Let’s do a filtering policy with a single FTP allowed rule ;



SOURCE	DESTINATION	VPN	SERVICE	ACTION	TRACK	INSTALL ON	TIME
* Any	* Any	* Any Traffic	TCP ftp	accept	- None	* Policy Targets	* Any

- And install the resulting policy.



Installation Process - Standard

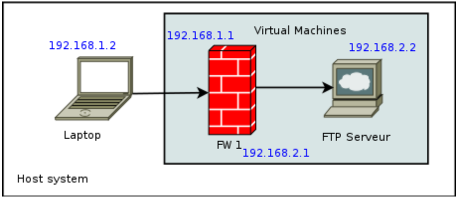
Installation

Installation Targets	Version	Network Security	IPS-1 Sensor	QoS	D
cpmodule	R75	Verification warning			

Progress

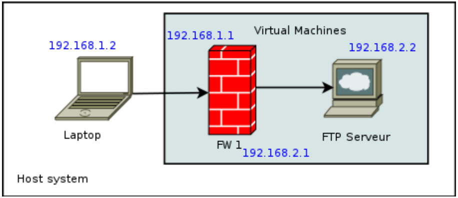
Installation completed with warnings !

Demonstration



Video

Demonstration



Video

Let’s have a firewall with a filtering policy allowing only port 21 and open a connection to port 22 on a FTP server.

Policy violation

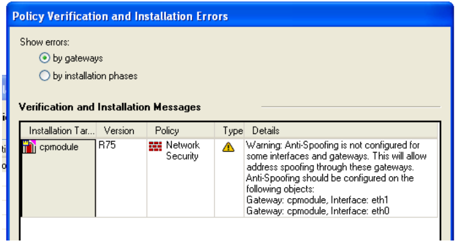
- One managed to open a connection to port 22
- With a filtering policy not allowing this

Policy violation

- One managed to open a connection to port 22
- With a filtering policy not allowing this
- But the connection was blocked after a few packets.

Policy violation

- One managed to open a connection to port 22
- With a filtering policy not allowing this
- But the connection was blocked after a few packets.
- Checkpoint GUI displays a warning about anti-spoofing.



Policy Verification and Installation Errors

Show errors:

● by gateways

○ by installation phases

Verification and Installation Messages

Installation Tar...	Version	Policy	Type	Details
cpmodule	R75	Network Security	⚠	Warning: Anti-Spoofing is not configured for some interfaces and gateways. This will allow address spoofing through these gateways. Anti-Spoofing should be configured on the following objects: Gateway: cpmodule, Interface: eth1 Gateway: cpmodule, Interface: eth0

There is no problem

Swift reaction of Checkpoint security team

> *Configuring anti-spoofing is a basic requirement.*
>
> *Them*

> *Are you planning some action regarding this issue?*
>
> *Me*

> *Anti-spoofing exists exactly for such issues. So [we] don't think that we need to do anything.*
>
> *Them*

There is no problem

Swift reaction of Checkpoint security team

> *Configuring anti-spoofing is a basic requirement.*
>
> *Them*

> *Are you planning some action regarding this issue?*
>
> *Me*

> *Anti-spoofing exists exactly for such issues. So [we] don't think that we need to do anything.*
>
> *Them*

Basic requirement

Choose your contractor well: the security level depends on his skills.

Others protocols

IRC

- As discussed before IRC helper provide the client with great power.
- The issue is inverted: can we act against client?
- Same technique applies with the following conditions:
  - Attacker and client are separated by firewall.
  - Attacker is on a network directly connected to the firewall.
  - IRC traffic can be sniffed by attacker (MITM or server).
- This is not interesting.

Others protocols

IRC

- As discussed before IRC helper provide the client with great power.
- The issue is inverted: can we act against client?
- Same technique applies with the following conditions:
  - Attacker and client are separated by firewall.
  - Attacker is on a network directly connected to the firewall.
  - IRC traffic can be sniffed by attacker (MITM or server).
- This is not interesting.

SIP

- The server sends port parameters in a similar way as FTP.
- The same attack is possible.
- Only the content has to be changed.

1. Introduction
   - Netfilter and the Conntrack
   - Degree of freedom in Netfilter helpers
2. Multi-layer attack
   - Conditions and principles
   - FTP case
   - Checkpoint
   - Others protocols
3. Impact and existing protection
   - Netfilter
   - Detecting the attack
4. Protocol analysis attack
   - Protocol analysis
   - Low TTL attack
   - The attack on nDPI and Suricata
5. Conclusion

Protection for Netfilter

- We only have to use the **rp_filter** feature.
- It is available since last century in all Linux kernel.

Protection for Netfilter

- We only have to use the rp_filter feature.
- It is available since last century in all Linux kernel.
- **Disabled by default**.

Protection for Netfilter

- We only have to use the rp_filter feature.
- It is available since last century in all Linux kernel.
- Disabled by default. Enabled by all decent firewall scripts.

Protection for Netfilter

- We only have to use the rp_filter feature.
- It is available since last century in all Linux kernel.
- Disabled by default. Enabled by all decent firewall scripts.
- To activate it:

```
echo "1"> /proc/sys/net/ipv4/conf/all/rp_filter
```

Protection for Netfilter

- We only have to use the rp_filter feature.
- It is available since last century in all Linux kernel.
- Disabled by default. Enabled by all decent firewall scripts.
- To activate it:

```
echo "1"> /proc/sys/net/ipv4/conf/all/rp_filter
```

- **Wait**

Protection for Netfilter

- We only have to use the rp_filter feature.
- It is available since last century in all Linux kernel.
- Disabled by default. Enabled by all decent firewall scripts.
- To activate it:

```
echo "1"> /proc/sys/net/ipv4/conf/all/rp_filter
```

- **Wait** and for IPv6?

Protection for Netfilter

- We only have to use the rp_filter feature.
- It is available since last century in all Linux kernel.
- Disabled by default. Enabled by all decent firewall scripts.
- To activate it:

```
echo "1"> /proc/sys/net/ipv4/conf/all/rp_filter
```

- **Wait** and for IPv6?
- No problem, let's set value in `/proc`:

```
echo "1"> /proc/sys/net/ipv6/conf/all/rp_filter
  /proc/sys/net/ipv6/conf/all/rp_filter: No such file or directory
```

Protection for Netfilter

- We only have to use the rp_filter feature.
- It is available since last century in all Linux kernel.
- Disabled by default. Enabled by all decent firewall scripts.
- To activate it:

```
echo "1"> /proc/sys/net/ipv4/conf/all/rp_filter
```

- **Wait** and for IPv6?
- No problem, let's set value in `/proc`:

```
echo "1"> /proc/sys/net/ipv6/conf/all/rp_filter
  /proc/sys/net/ipv6/conf/all/rp_filter: No such file or directory
```

IPv6 protection for Netfilter

Since 3.3

- Can use Netfilter rpfilter module by Florian Westphal

```
iptables -A PREROUTING -t raw \
  -m rpfilter --invert -j DROP
```

- PREROUTING raw is before all Netfilter treatment

IPv6 protection for Netfilter

Since 3.3

- Can use Netfilter rpfilter module by Florian Westphal

```
iptables -A PREROUTING -t raw \
  -m rpfilter --invert -j DROP
```

- PREROUTING raw is before all Netfilter treatment

Before 3.3

- A manual setup is needed.
- Dedicated ip6tables rules need to be written.
- The network topology needs to be known.
- Good implementations already implement these rules.
- Some were doing it badly.

IPv6 protection for Netfilter

The bad ruleset

```
ip6tables –A FORWARD –m state ––state ESTABLISHED,RELATED –j ACCEPT
ip6tables –A FORWARD –i $CLIENT_IFACE ! –s $CLIENT_NET –j DROP
```

- The attack packet is valid for Netfilter.
- It belongs to an established connection.
- It is accepted by the first rule and never reaches the anti-spoofing rule.

IPv6 protection for Netfilter

The bad ruleset

```
ip6tables –A FORWARD –m state ––state ESTABLISHED,RELATED –j ACCEPT
ip6tables –A FORWARD –i $CLIENT_IFACE ! –s $CLIENT_NET –j DROP
```

- The attack packet is valid for Netfilter.
- It belongs to an established connection.
- It is accepted by the first rule and never reaches the anti-spoofing rule.

The good ruleset

```
ip6tables –A PREROUTING –t raw –i $CLIENT_IFACE ! –s $CLIENT_NET –j DROP
```

- Raw table is before the FORWARD chain and even before connection tracking related operations.
- The packet is dropped before causing any problem.

Suricata Rules

- Largely compatible with Snort syntax
- Able to use VRT and Emerging Threats rulesets

```
alert tcp any any -> 192.168.1.0/24 21 (content: "USER root"; msg: "FTP root login";)
```

Suricata Rules

- Largely compatible with Snort syntax
- Able to use VRT and Emerging Threats rulesets

```
alert tcp any any -> 192.168.1.0/24 21 (content: "USER root"; msg: "FTP root login";)
```

Action: alert / drop / pass

Suricata Rules

- Largely compatible with Snort syntax
- Able to use VRT and Emerging Threats rulesets

```
alert tcp any any -> 192.168.1.0/24 21 (content: "USER root"; msg: "FTP root login";)
```

IP Parameters

Suricata Rules

- Largely compatible with Snort syntax
- Able to use VRT and Emerging Threats rulesets

```
alert tcp any any -> 192.168.1.0/24 21 (content: "USER root"; msg: "FTP root login";)
```

Pattern

Suricata Rules

- Largely compatible with Snort syntax
- Able to use VRT and Emerging Threats rulesets

```
alert tcp any any -> 192.168.1.0/24 21 (content: "USER root"; msg: "FTP root login";)
```

Other parameters

Counter measures on Suricata

- FTP injection attack has consequences, TCP data is injected
- Server doesn’t know, so sends data for the same sequence number
- Resultsing in overlapping data, which is different

Detecting overlapping data

- TCP Stream reassembly engine detects this and sets an event
- Rule keyword "stream-event":

Stream-event

```
stream-event:reassembly_overlap_different_data;
```

Detecting overlapping data

- Attack is pretending to come from server.
- Full example rule:

Rule

```
alert tcp any 21 -> any any (msg:"Overlap data";    \
  flow:to_client; dsize:>0;                         \
  stream-event:reassembly_overlap_different_data;   \
  classtype:protocol-command-decode; sid:1; rev:1;)
```

Detect FTP injection

- FTP attack uses unsollicited 227 response to fool the helper
- Normally a 227 follows a PASV command
- We can detect this using rules

Detect FTP injection - step 1

- Detect PASV and set flowbit
- No alert, as this is common and benign

Rule 1

```
alert tcp any any -> any 21 (msg:"FTP PASV cmd";        \
  flow:to_server; content:"PASV"; depth:4;              \
  flowbits:set,ftp.pasv_seen;  noalert;                 \
  classtype:not-suspicious; sid:1; rev:1;)
```

Detect FTP injection - step 2

- Detect 227 response and see if PASV was seen before

Rule 2

```
alert tcp any 21 -> any any                                        \
  (msg:"FTP unsolicited 227, possible injection";                  \
  flow:to_client; content:"227"; depth:3;                          \
  flowbits:isnotset,ftp.pasv_seen;                                 \
  flowbits:set,ftp.possible_injection; noalert;                    \
  classtype:protocol-command-decode; sid:2; rev:1;)
```

- We could already alert here, but taking it one step further

Detect FTP injection - step 3

- We already know we have a unsollicited 227
- Now combine it with stream event

Rule 3

```
alert tcp any 21 -> any any                                        \
  (msg:"FTP PASV 227 injection attack";                            \
  flow:to_client;                                                  \
  flowbits:isset,ftp.possible_injection;                           \
  stream-event:reassembly_overlap_different_data;                  \
  classtype:misc-attack; sid:3; rev:1;)
```

Detect FTP injection - port

- Attack uses injected 227 response to punch hole
- Attacker cares about non-FTP ports maybe
- We can detect the port the attacker wants

Detect FTP injection - port

- Injected 227 contains port to use
- Syntax: "227 Entering Passive Mode (192,168,2,2,12,234)"
- Port is calculated, 1st port value * 256 + 2nd value
- Because of calculation, pcre is limited use

Rule

```
alert tcp any 21 -> any any                                   \
  (msg:"FTP 227 to privileged port";                          \
  flow:to_client; content:"227"; depth:3;                     \
  pcre:"/^227\s[A-z\s]+\((\d+,){4}0,/m";                      \
  pcre:"/(?!2[0-1]\))/R";                                     \
  classtype:protocol-command-decode; sid:7; rev:1;)
```

- Similary we can detect other ports, like MySQL 3306 port

Luajit keyword

- We can also use experimental luajit keyword
- This allows for Lua script to be called
- luajit support is currently in beta
- This way we can calculate the actual port value

Rule

```
alert tcp any 21 –> any any                                        \
  (msg:"FTP 227 to restricted port";                               \
  flow:to_client; content:"227"; depth:3;                          \
  luajit:hack_lu.lua;                                              \
  classtype:protocol–command–decode; sid:8; rev:1;)
```

Luajit script

Simplified Script

```
function match(args)
    a = tostring(args["payload"])
    if #a > 0 then
        if a:find("^227") then
            for str,str2 in a:gmatch("227 Entering Passive Mode " \
                            "%(%d+,%d+,%d+,%d+,(%d+),(%d+)%)") do
                port = tonumber(str) * 256 + tonumber(str2)
                if port < 1024 and port ~= 20 and port ~= 21 then
                    return 1
                elseif (port == 3306) then
                    return 1
                else
                    return 0
                end
            end
        end
    end
    return 0
end
```

1 Introduction
- Netfilter and the Conntrack
- Degree of freedom in Netfilter helpers

2 Multi-layer attack
- Conditions and principles
- FTP case
- Checkpoint
- Others protocols

3 Impact and existing protection
- Netfilter
- Detecting the attack

4 Protocol analysis attack
- Protocol analysis
- Low TTL attack
- The attack on nDPI and Suricata

5 Conclusion

Protocol analysis: a difficult task

High performance system

- Protocol analysis mean high speed parsing
- Make the task that hundred of clients does

Two steps algorithm

- Classification
  - Analyse the trafic to detect pattern corresponding to a know protocol
  - Decide which protocol is used
- Decoding
  - Parse packet following protocol specification

Limit the cost of classification

- Classification means looking for patterns for all protocols
- Once it is done the protocol is assumed to be fix

Available products

From recognition to protocol decoding

- Protocol recognition: nDPI, l7filter, NG-firewall
- Some decoded protocols: Suricata
- Decoding: Qosmos

Business as usual

- A lot of high profit applications
  - From Network Security Monitoring
  - To user behavior control
- A lot of work to maintain applications
- Few opensource implementation with a long list of protocols
  - l7-filter
  - nDPI

Example of nDPI

History

- Originally called OpenDPI
- Released under GPL by Ipoque
- Closed source after Ipoque has been bought
- Forked by Luca Deri and Ntop team under nDPI name

Description

- A C library implementating protocol recognition
- More than 100 supported protocols:
  - HTTP, Google, MSsql, Worl of Kung Fu, . . .
- Library is used for
  - Sniffing in Ntop
  - Packet filtering in Netfilter

Protocol detection implementation in Suricata

- Based on fixed strings currently, e.g. "GET " for HTTP
- "probing parser" parses protocol to verify
- then hands off TCP connection to real parser
- Protocol detection runs on top of TCP stream reasssembly

Causing recognition mistake

Evading classification

- If the protocol is not recognized, it can’t be decoded
- Classification evasion lead to undetected traffic

Classification made easy

- Some protocol can be classified with a single message
- Sending packet with fake content prior to real one
- Will lead to recognition mistake

Don’t mess with the server

Issue when sending fake content

- A standard server will not understand protocol change
- Client connection risk to be closed
- Need to avoid to send fake content to server

Don’t mess with the server

Issue when sending fake content

- A standard server will not understand protocol change
- Client connection risk to be closed
- Need to avoid to send fake content to server

Using low TTL value

- Bad packet must die before reaching server
- Using low TTL value can do the trick
- Best TTL for that is one less than distance to server

computing TTL value

TTL choice is arbitrary

- No RFC and a list of choice made for each OS
- The value for major OS is 128 or 64

Computation of TTL using distance to server

```
if $TTL_S >= 64$ then
    $\Delta \leftarrow 128 - TTL_S$
else
    $\Delta \leftarrow 64 - TTL_S$
end if
$TTL_{attack} \leftarrow \Delta - 1$
```

Implementing the attack in opensvp

Sniffing is not enough

- Regular traffic need to be blocked
- Before we inject the attack packets

Netfilter to the rescue

- Block the packet with NFQUEUE
- Get the packet in userspace
- Send forged packet
- Release blocked packet

Nfqueue-bindings power

- Python binding for libnetfilter_queue
- Multi language and easy access to NFQUEUE features
- Available at
  `https://www.wzdftpd.net/redmine/projects/nfqueue-bindings/wiki/`

Attack on nDPI

Test used

- Injection of packet with HTTP header during capture with opensvp

  ```
  opensvp -q 0 -i eth0 -n
  ```

- SMTP traffic is targeted by the attack
- A pcap is captured with and without opensvp running

Attack on nDPI

Test used

- Injection of packet with HTTP header during capture with opensvp

  ```
  opensvp -q 0 -i eth0 -n
  ```

- SMTP traffic is targeted by the attack
- A pcap is captured with and without opensvp running

Result of analysis with nDPI pcap reader

- Plain pcap analysis
  - unknown: 7 packets
  - Mail_SMTP: 8 packets
- Pcap with attack
  - unknown: 17 packets

Impact of the attack on Suricata

- Suricata is currently unaware of network topology with regard to TTL
- So it isn't able to "know" if a packet's TTL is too low to reach the host
- Still, very low TTL is unusual
- We can create a rule for that

Detecting very low TTL

- Example TTL rule for TCP data packets with very low TTL

TTL rule

```
alert tcp any 21 -> any any \
  (msg:"TCP data pkt with low TTL"; \
  dsize:>0; ttl:<10; sid:1; rev:1;)
```

Limitations with protocol recognition

- Several protocols use very similar structures
- e.g. IRC, SMTP, FTP
- Suricata currently doesn’t support this

Future changes in Suricata

- We're currently rewriting protocol detection
- More aggressive use of "probing parsers"
- Make it easier to support protocols like SMTP, IRC, FTP properly
- Also adding a high level protocol keyword, allowing for "port 25 and NOT smtp"

1. Introduction
   - Netfilter and the Conntrack
   - Degree of freedom in Netfilter helpers
2. Multi-layer attack
   - Conditions and principles
   - FTP case
   - Checkpoint
   - Others protocols
3. Impact and existing protection
   - Netfilter
   - Detecting the attack
4. Protocol analysis attack
   - Protocol analysis
   - Low TTL attack
   - The attack on nDPI and Suricata
5. Conclusion

Using low layer to attack

Low layer attack are still working

- rp_filter was not implemented for IPv6 for years
- Checkpoint default setup is non secure
- TTL can’t be followed easily

Using low layer to attack

Low layer attack are still working

- rp_filter was not implemented for IPv6 for years
- Checkpoint default setup is non secure
- TTL can’t be followed easily

And will work for long

- Checkpoint default setup will not change
- Implementation of rp_filter in Netfilter will not guarantee it is widely used
- Mobility will increase TTL volatility
- A necessary trade off between performance and security
  - Real time and high bandwidth force equipments to approximation
  - A centralized equipment can’t impersonate all the internet

SNMP: Security is Not My Problem

Kernel developers are full disclosure advocates

> *Security issues are just bugs, and we report bugs on the public mailing list and try to fix them.*
>
> *A Linux kernel developer*

SNMP: Security is Not My Problem

Kernel developers are full disclosure advocates

> *Security issues are just bugs, and we report bugs on the public mailing list and try to fix them.*
>
> *A Linux kernel developer*

Secure default is not vendor problem

> *Anti-spoofing exists exactly for such issues. So [we] don't think that we need to do anything.*
>
> *Checkpoint security team*

Questions

Do you have any questions?

Thanks to

- Pablo Neira, Patrick McHardy: Netfilter developers are cool
- Florian Westphal: for implementing Netfilter-based RP filter

More information

- Secure use of Iptables and connection tracking helpers: `http://home.regit.org/netfilter-en/secure-use-of-helpers/`
- Victor's blog : `http://www.inliniac.net`
- Eric's blog : `https://home.regit.org`

Contact us

- Eric Leblond: eric@regit.org, @Regiteric on twitter
- Victor Julien: victor@inliniac.net, @inliniac on twitter

Degree of freedom of helpers

Module	Source	Port Source	Destination	Port Dest	Proto	Option
amanda	Fixed	0-65535	Fixed	In CMD	TCP	
ftp	Fixed	0-65535	In CMD	In CMD	TCP	loose = 1 (dflt)
ftp	Full	0-65535	In CMD	In CMD	TCP	loose = 0
h323	Fixed	0-65535	Fixed	In CMD	UDP	
h323 q931	Fixed	0-65535	In CMD	In CMD	UDP	
irc	Full	0-65535	Fixed	In CMD	TCP	
netbios_ns	Iface Network	Fixed	Fixed	Fixed	UDP	
pptp	Fixed	In CMD	Fixed	In CMD	GRE	
sane	Fixed	0-65535	Fixed	In CMD	TCP	
sip rtp_rtcp	Fixed	0-65535	Fixed	In CMD	UDP	sid_direct_media = 1 (dflt)
sip rtp_rtcp	Full	0-65535	In CMD	In CMD	UDP	sid_direct_media = 0
sip signalling	Fixed	0-65535	Fixed	In CMD	In CMD	sip_direct_signalling = 1 (dflt)
sip signalling	Full	0-65535	In CMD	In CMD	In CMD	sip_direct_signalling = 0
tftp	Fixed	0-65535	Fixed	In Packet	UDP	