

# Argos Logs Explained

## 1. General Information

The logs produced by Argos contain the memory fingerprint of a detected attack. All tainted system state is exported. Argos isolates the attacked process by only looking up memory pages accessible to it. This way the volume of exported data is reduced to exclude data that cannot be associated with the attack. Furthermore, when provided with a hint about the guest operating system (please refer to the ARGOS (1) manpage), it is able to distinguish between user and kernel memory space further reducing the volume exported data.

Logs are written to a file in the working directory following the naming format shown below (*rid* is a randomly generated integer ID identifying the detected attack):

```
argos.csi.[rid]
```

## 2. Logs Format

All Argos logs follow a strict format that begins with a log header followed by multiple blocks of memory contents, each block preceded by a memory block header. The end of the log is marked by an empty memory block header.

LOG HEADER	
MEMORY BLOCK HEADER	MEMORY BLOCK CONTENTS
....	
EMPTY MEMORY BLOCK HEADER	

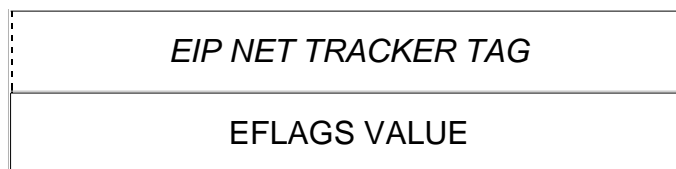
**Log high-level format**

Depending on the version of Argos log formats might differ. In the rest of this document each version will be explained in detail.

### 2.1 Log header

FORMAT	ARCH	TYPE	TIMESTAMP
REGISTER VALUES			
TAGS VALUES			
<i>NET TRACKER VALUES</i>			
EIP VALUE		EIP TAG	

**Log header format**



The header always starts with the version number that determines the format of the rest of the header and log. The second field specifies the emulated architecture that produced the log. The architecture field also affects the rest of the data in the header and log. The following fields specify the type of the alert that generated the log, and the time it was issued.

The rest of the header's fields are dependant upon the architecture. First the contents of the general purpose registers. *x86* architectures have 8 32 bit registers, while *x86\_64* architectures have 16 64 bit registers. The sequence the registers are written out is the following: EAX, ECX, EDX, EBX, ESP, EBP, ESI and last EDI.

Register contents are followed by their corresponding tags. Each tag indicates two things: one whether the register contents are tainted, and two (if it is tainted) where was this value loaded from in physical memory. There is one tag for each general purpose register and they have the same size. If the value of the tag is non-zero the contents of the corresponding register are tainted, and corresponds to the physical memory address where the tainted data were read from. If the register contents have been directly read by the network, the value of the tag is -1 (0xffffffff).

After the general purpose registers we find the instruction pointer (EIP) along with its tag. Normally, the tag for EIP should be tainted for an alert to be issued, and EIP should contain a memory address provided by the attacker (usually the address where shellcode was injected).

Finally, the last field of the header is the value of the EFLAGS register (size also depends on architecture).

The following table summarises the header's fields:

Name	Size	Description
FORMAT	1 byte	Format of header. Bit 8: Log contains net tracker data. Bit 7: specifies that host endianness is big endian. Bits 6 - 1: Specify header version number.
ARCH	1 byte	Architecture emulated. 0: I386. 1: X86_64.
TYPE	2 bytes (host endianness)	Type of attack is one the <b>ALERT TYPES</b> shown in the table at the end of this document.
TIMESTAMP	4 bytes (host endianness)	32 bit timestamp of the attack.
REGISTER VALUES	Depends on architecture <sup>1</sup> (Host endianness)	The values of the architecture's registers.

#### Log header fields specification

TAG VALUES	Depends on architecture <sup>1</sup> (Host endianness)	Determines whether a register is tainted. When not zero, it also represents where from physical memory the register's contents were loaded from. Zero if the contents of the register are not tainted.
NET TRACKER VALUES	A 4 byte value for each register (host endianness)	Only present when bit 8 of <i>FORMAT</i> is asserted. Invalid if the tag of a register is zero. For tainted registers contains the index of the register's data within the network trace logged by Argos. <sup>2</sup>
EFLAGS	Depends on architecture (architecture endianness)	The value of the EFLAGS register.

<sup>1</sup>Registers and tags are 32 bit long on I386 and 64 bit long on x86\_64.

<sup>2</sup>Tracking of network values is lossy. This implies that the actual location of memory data in the network trace can slightly diverge from the index provided. In the case that 2 or more network values are combined to create a new value, the network index provides information only for the latest network value used in the calculation. Furthermore, the exact location of the data could be in a 32 byte window (the centre of the window is provided by the network index).

For reference, the C structure used within Argos is shown below (target\_ulong in an unsigned long at the emulated architecture's level and CPU\_NB\_REGS is the architecture's number of general purpose registers):

```
struct argos_log_hdr_struct {
    uint8_t format;
    uint8_t arch;
    uint16_t type;
    uint32_t ts;
    target_ulong reg[CPU_NB_REGS];
    target_ulong rtag[CPU_NB_REGS];
    target_ulong eip,
                eiptag;
    target_ulong eflags;
} __attribute__((packed));
```

## 2.2 Memory block header

The header always starts with the version number that determines the format of the rest of the header. The second field is a flag that specifies whether the memory block is tainted or not. For most of the blocks this flag is asserted, there is the case though that EIP points to a memory area that is not tainted. In that case the memory block is written to the log and the taint-ness flag is set to zero. The third field specifies the size of the block's contents that follows the header.

FORMAT	TAINTED	SIZE
PADDR		VADDR

**Memory block header version 1 format**

The memory address of the block is next. Both the physical and virtual address of the block are written out. If a certain memory block (or page if you prefer) is mapped to multiple virtual

memory addresses then the block will be written out multiple times. The memory block's contents are in the architecture's endianness.

If net tracker data are present they follow the block's contents. For every byte of memory a 4 byte integer follows that points to the origin of that byte within the network trace log.

Field name	Length (Bytes)	Description
FORMAT	1	Format version. Bit 8: Block contains net tracker data. Bits 6 - 1: Specify header version number
TAINTED	1	Taint-ness flag.
SIZE	2 (Host endianness)	Size of block's contents following the header.
PADDR	4(I386) or 8(X86_64) (Host endianness)	Physical memory address of block.
VADDR	4(I386) or 8(X86_64) (Host endianness)	The values of the 8(I386) or 16(X86_64) registers.

**Virtual memory address of block.**

For reference, the C structure used within Argos is shown below (target\_ulong in an unsigned long at the emulated architecture's level):

```
struct argos_mblock_hdr_struct {
    uint8_t format;
    uint8_t tainted;
    uint16_t size;
    target_ulong paddr;
    target_ulong vaddr;
} __attribute__((packed));
```

Name	Value	Description
ARGOS_ALERT_JMP	0	A tainted was used in an indirect jmp instruction.
ARGOS_ALERT_LJMP	1	A tainted value was used in a protected mode jmp instruction. [OBSOLETE] Merged with indirect jmp.
ARGOS_ALERT_TSS	2	A tainted value was used to load EIP within TSS.
ARGOS_ALERT_LCALL	3	A tainted value was used in a real mode call instruction.
ARGOS_ALERT_IRET	4	A tainted value was used in an iret instruction.
ARGOS_ALERT_RET	5	A tainted value was used in a ret instruction.
ARGOS_ALERT_WRMSR	6	A tainted value was written in an MSR register. [OBSOLETE] Will be completely removed.
ARGOS_ALERT_CI	7	Execution flow was redirected to an instruction that is tainted.

#### Alert Types

### 3. Network Trace Log Format

When the network tracker mode is enabled, argos will also log all the network data received do a trace file. The format of this file is shown below:

Description	Size
ETHERNET FRAME SIZE	2 bytes (Little endian)
ETHERNET FRAME DATA	Specified by the ethernet frame size

#### Net tracker log format