

# Next-Generation Debuggers

For Reverse Engineering

**The ERESI team**

**[eresi@asgardlabs.org](mailto:eresi@asgardlabs.org)**



**Black Hat Briefings**

# This presentation is about ..

- The Embedded ERESI debugger : e2dbg
- The Embedded ERESI tracer : etrace
- The ERESI reverse engineering language
- Unification & reconstruction of debug formats
- Program analysis builtins (focusing on control flow graphs)



# The ERESI project

- Started in 2001 with the ELF shell
- Developed at LSE (EPITA security laboratory)
- Contains more than 10 components
- Featured in 2 articles in Phrack Magazine:
  - The Cerberus ELF Interface (2003)
  - Embedded ELF Debugging (2005)



# Limitations of existing UNIX debugging framework

- GDB : Use OS-level debugging API (ptrace) -  
> does not work if ptrace is disabled or absent
- Very sensible to variation of the environment  
(ex: ET\_DYN linking of hardened gentoo)
- Strace / Ltrace : use ptrace as well. Very few  
interaction (command-line parameters)



# Limitations of existing frameworks

- **None of these frameworks rely on a real reverse engineering language**



# The ERESI team

- Started with a single person in 2001 (The ELF shell crew). Remained as it during 3 years.
- Another person developed libasm (disassembling library) since 2002
- A third person developed libdump (the network accessibility library) in 2004-2005
- Since mid-2006 : community project (6 persons)



# The modern ERESI project

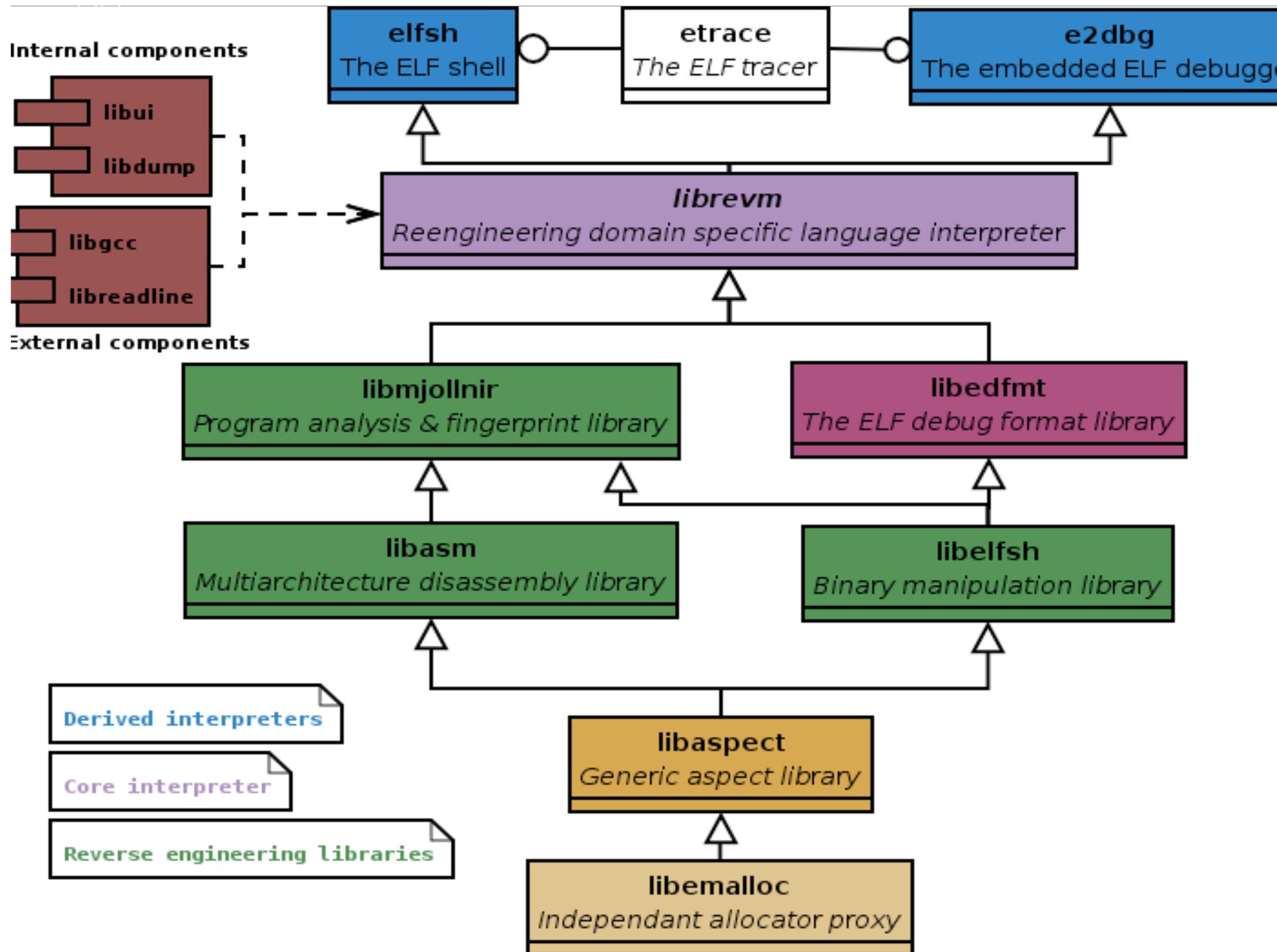
- elfsh (and libelfsh): The ELF shell
- **e2dbg** (and libe2dbg): The Embedded ELF debugger
- **etrace** : The Embedded tracer
- librevm : the language interpreter
- libmjollnir : fingerprinting & graphs library
- libaspect : Aspect oriented library



# The modern ERESI project (cont)

- **libasm** : typed disassembling library
- **libedfmt** : the ERESI debug format library
- **liballocproxy** : allocation proxying library
- **libui** : The user interface (readline-based)





# ERESI contributions (1)

- **Can debug hardened systems** (does not need ptrace) : PaX/grsec compatible
- **Very effective analysis** : improve the performance of fuzzing, heavy-weight debugging (no context switching between the debugger and the debuggee : the dbgvm resides in the debuggee)



## ERESI contributions (2)

- A **reflective framework** : possibility to change part of it in runtime without recompilation
- **The first real reverse engineering language !!!**
  - hash tables
  - regular expressions
  - loops, conditionals, variables
  - The complete ELF format objects accessible from the language



# The ERESI language : example 1

```
load /usr/bin/ssh
```

```
set $entnbr 1.sht[.dynsym].size
```

```
div $entnbr 1.sht[.dynsym].entsize
```

```
print Third loop until $entnbr :
```

```
foreach $idx of 0 until $entnbr
```

```
  print Symbol $idx is 1.dynsym[$idx].name
```

```
forend
```

```
unload /usr/bin/ssh
```



# The ERESI language :

## example 2

```
add $hash[hname] Intel
add $hash[hname] Alpha
add $hash[hname] Sparc32
add $hash[hname] Mips
add $hash[hname] Sparc64
add $hash[hname] AMD
add $hash[hname] Pa-risc
foreach $elem of hname matching Sparc
    print Regex Matched $elem
endfor
```



# List of available hash tables

- Basic blocks (key: address)
- Functions (key: address)
- Regular expression applied on the key
- Many dozen of hash tables (commands, objects ..) : see **tables** command of ERESI
- Currently not supported : hash table of instructions, of data nodes (too many elements) => need of demand-driven analysis



# The ERESI language :

## example 3

```
type archtypes = elm:string[55]
inform archtypes elfsh_arch_type
type archaddr = elm:long[55]
inform archaddr elfsh_arch_type
print Now print Strings
print 107.archtypes[elfsh_arch_type].elm[0]
print 107.archtypes[elfsh_arch_type].elm[1]
print Now print addresses
print 107.archaddr[elfsh_arch_type].elm[0]
print 107.archaddr[elfsh_arch_type].elm[1]
```

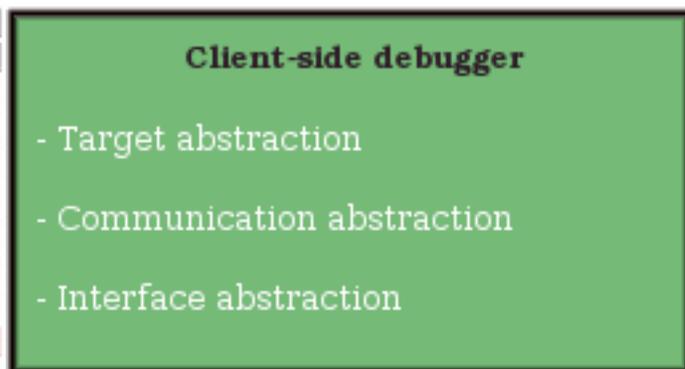


# e2dbg : the Embedded ELF debugger

- Does not use ptrace. Does not have to use any OS level debug API. Evades PaX and grsecurity.
- Proof of concept developed on Linux / x86 .
- Scriptable using the ERESI language
- Support debugging of multithreads
- No need of **ANY** kernel level code (can execute in hostile environment)



ERESI interpreter = **Embedded debugger**  
+ Unintrusive heap  
+ analysis code  
+ aspect library  
+ debug format handling



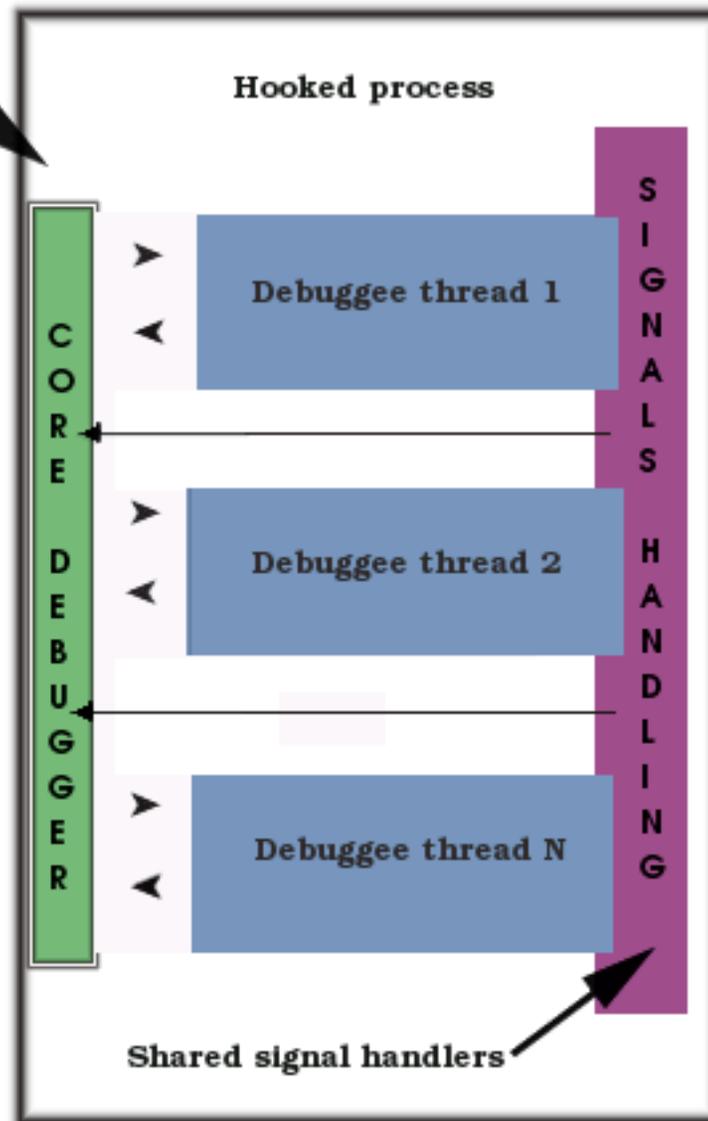
FIFO  
INET  
(...)



→ Signals

↔ Interprocess communication

➤ Intraprocess communication



# e2dbg : features

- **Classical features:**
  - breakpoints (using processor opcode or function redirection)
  - stepping (using sigaction() syscall)
- **Allocation proxying**
  - keep stack and heap unintrusiveness
- **Support for multithreading**



# Allocation proxying

- We manage **two different heap allocators** in a single process:

```
int hook_malloc(int sz)
{
    if (debugger)
        return (aproxy_malloc(sz));
    return (orig_malloc(sz))
}
```



# Handling of debug format & The Embedded ELF Tracer (etrace)



# Debugging format

- Describe each element of a program
  - Give names and position of:
    - Variables
    - Functions
    - Files
    - ....
  - Store program types dependences between them



# Debugging format - issues

- Distinction of debugging format
  - stabs, dwarf, stabs+, dwarf2, gdb, vms ...
  - Different ways to parse, read, store ...
- For example with stabs and dwarf2
  - Stabs does not contain any position reference
    - You store the whole parsing tree
  - Dwarf2 use read pattern apply directly on data
    - You cannot store everything (too big)
  - ...



# Uniform debugging format

- **Parsing**
  - So we can read the debugging format
- **Transforming**
  - We transform it on a uniform representation
  - Keep only useful information
- **Cleaning**
  - We keep only uniform debugging format
- **New debugging format**
  - We change only backend part
- **Register types on ERESI type engine**



# Embedded ELF tracer

- Tracer using ELFsh framework
- Tracing internal and external calls
- Dynamic and supports multiple architecture
  - It does not use statically stored function prototypes
  - Use gcc to reduce architecture dependence
- Work with and without debugging format
- Recognize string, pointers and value



# Embedded ELF trace - script

```
#!/usr/local/bin/elfsh32
load ./sshd
traces add packet_get_string
traces create privilege_sep
traces add execv privilege_sep
traces create password
traces add auth_password password
traces add sys_auth_passwd password
save sshd2
```



# Etrace – output on sshd

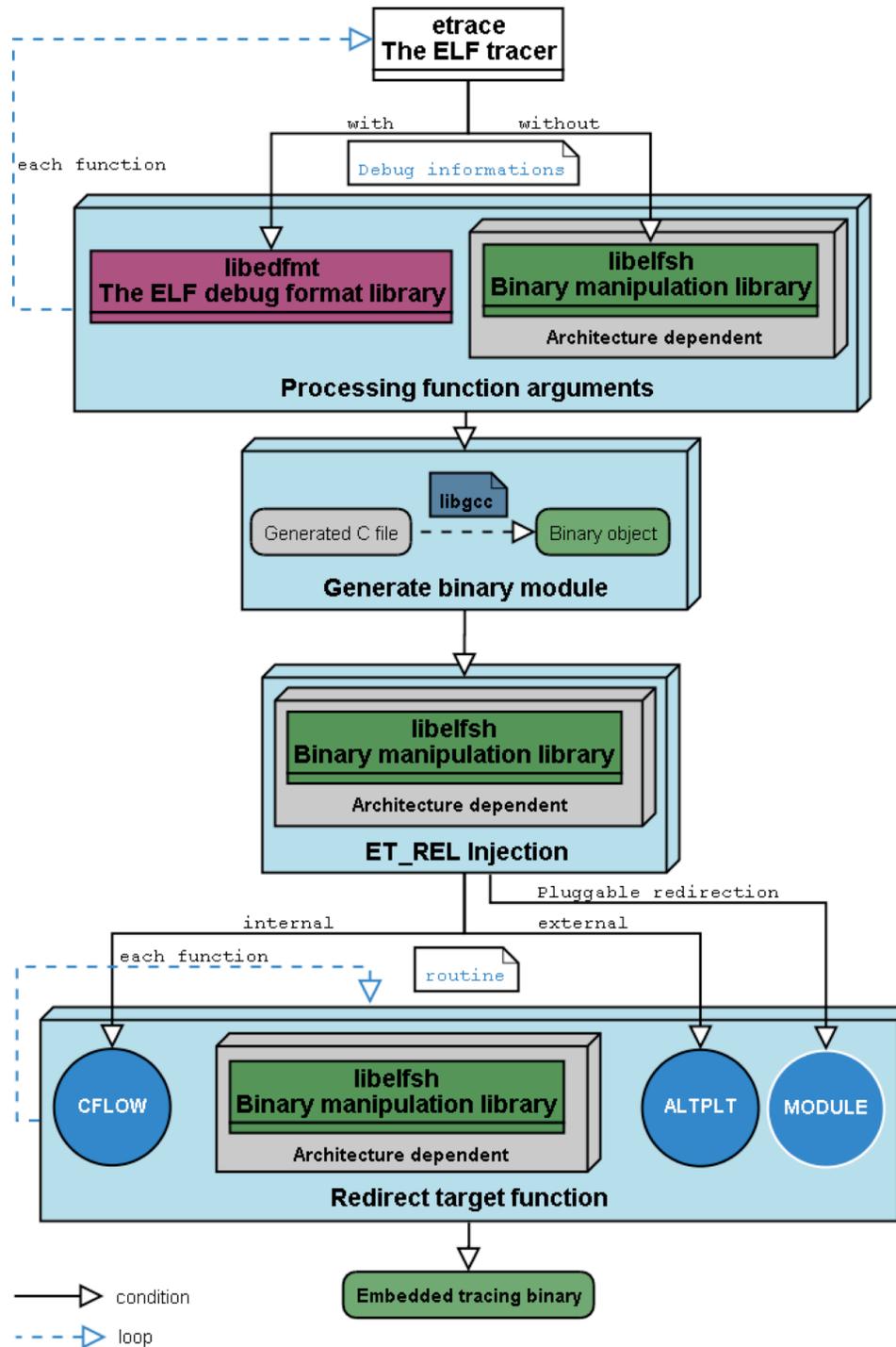
```
+ execv(*0x80a5048 "(...)/openssh-4.5p1/sshd2", *0x80aa0a0)
+ packet_get_string(*u_int length_ptr: *0xbf8f4738)
- packet_get_string = *0x80ab9f0 "mxatone"
debug1: Attempting authentication for mxatone. (...)
+ packet_get_string(*u_int length_ptr: *0xbf8f42fc)
- packet_get_string = *0x80a9970 "test1"
+ auth_password(*Authctxt authctxt: *0x80aaca0, void* password:
*0x80b23a8 "test1")
+ sys_auth_passwd(*Authctxt authctxt: *0x80aaca0, void*
password: *0x80b23a8 "test1")
- sys_auth_passwd = 0x0
- auth_password = 0x0
```



# Etrace – Performance

function name	etrace (sec)	ltrace (sec)	ratio
open	0.000072	0.000106	1.47
write	0.000070	0.000106	1.51
crypt	0.001560	0.001618	1.03
calloc	0.000143	0.000200	1.39
unlink	0.000046	0.000082	1.78
puts	0.000033	0.000078	2.36
getcwd	0.000009	0.000039	4.33
close	0.000007	0.000038	5.42
strdup	0.000007	0.000022	3.14
free	0.000005	0.000020	4.00





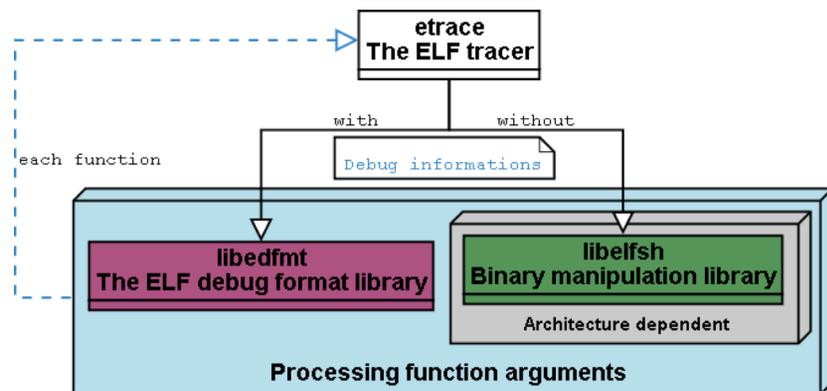
# Embedded ELF tracer

- **Trace backend**
  - Analyze target function
  - Create proxy functions
- **Embedded tracer**
  - Inject proxy functions in the binary
  - Redirect calls into our proxy functions
  - Create a new binary
- **Automatic using the ELF tracer**



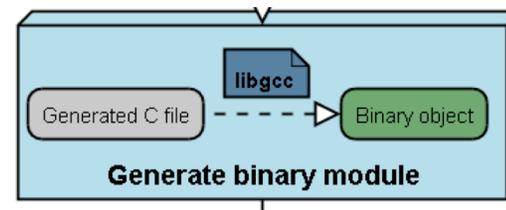
# Etrace - Processing function arguments

- With debugging information
  - Extract arguments information
    - size
    - names
    - type names
    - ...
- With architecture dependent argument counting
  - Backward analysis
  - Forward analysis



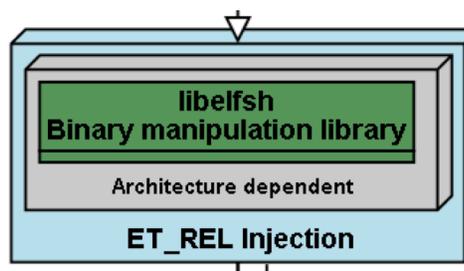
# Etrace - Generate binary module

- **Generate a .c file**
  - Call tree (padding)
  - Dynamic check pointers, strings or value
- **Benefits**
  - Architecture independent
  - New feature implementation
  - Less bugs
  - Use ELFsh framework



# Libelfsh - ET\_REL injection

- **ET\_REL injection principle**
  - Add a binary module directly on target binary
- **Merge symbols and sections list**
- **Section injection**
  - Code sections
    - Injected before .interp
  - Data sections
    - Injected after .bss

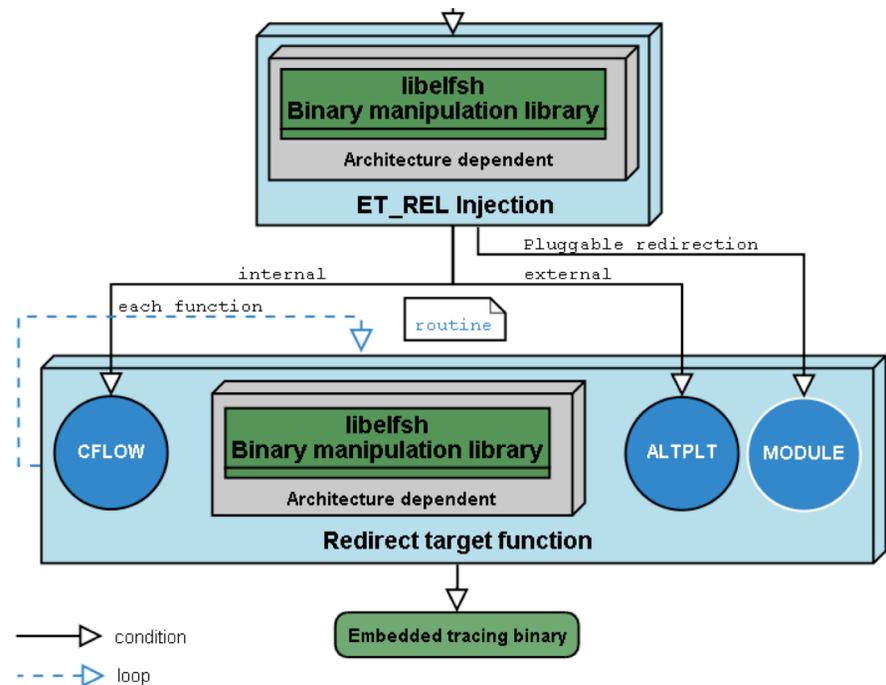


**Relocation in two steps**



# Libelfsh - Redirect target function

- **Internal function**
  - CFLOW technique
- **External function**
  - ALTPLT technique
- **Custom redirection**
  - Vector benefit
  - Your own redirection mechanism



# Program analysis



**Black Hat Briefings**

# A Graph Analyzer

- **Graph analyzers**
  - Identify blocks and functions
  - Identify links (calls and jumps)
  - Build a graph with this info
- **Control Flow Graphs (CFGs)**
  - Inter-blocks CFGs vs. Interprocedural CFGs
  - Main instrument to Control Flow analysis



# A Graph Analyzer

- **Control Flow Analysis**
  - Essential to some kinds of further analysis and to optimization
  - Gives information about properties such as
    - Reachability
    - Dominance
    - ...



# A Graph Analyzer – Libasm

- **Libasm**
  - Lowest layer of this application
  - Multi-architecture disassembling library
    - Intel IA-32
    - SPARC V9
    - In the near future, MIPS
  - Unified type system



# A Graph Analyzer – Libasm

Type	Description
IMPBRANCH	Imperative branch (jump)
CONDBRANCH	Conditional branch
CALLPROC	Call to a procedure
RETPROC	Return from a procedure
ARITH	Arithmetic or logic operations
LOAD	Memory data load
STORE	Memory data store
ARCH	Architecture-dependent instruction
FLAG	Flag-modifier instruction
INT	Interrupt or call-gate instruction
ASSIGN	Assignment instruction
TEST	Comparison or test instruction
NONE	Instruction that doesn't fit any of the above



# A Graph Analyzer – Libasm

- The unified instruction type system
  - Works with non-mutually exclusive types
  - Provides means to “blindly” analyze an instruction
  - Eg. Control Flow analysis!



# A Graph Analyzer - Libasm

- **Libasm vectors**
  - Storage of pointers to opcode handling functions
  - 4 dimensions: 1 for machine info, 3 for opcode info
  - Runtime dumping and replacing of vectors
    - Built-in language constructs
    - Easy-made opcode tracer!



# A Graph Analyzer – libmjollnir

- **Libmjollnir**
  - Upper-layer component
  - Code fingerprinting and program analysis
- **CFG construction**
  - Libmjollnir treats both: blocks and functions
  - Separate representations (structures)



# A Graph Analyzer – libmjollnir

- **Containers**
  - Generic structures to encapsulate blocks and functions
  - Have linking (input and output links) information
  - Have a pointer to data and type information to interpret this data accordingly



# A Graph Analyzer – libmjollnir

- **Containers**
  - Allow for more abstract graph analysis (analyzing a graph of containers)
  - In the future, may also store data nodes (Data Flow analysis)
  - Also for the future, containers of containers
    - Even higher abstraction of links and relationships



# Conclusion



# Conclusion

- New foundations for reverse engineering and debugging of closed-source software using in-process analysis
- A language approach for reversing
- Many concrete applications (embedded tracer and debugger)



# The near future

- Binding of demand-driven dataflow analysis in the ERESI language
- Program transformation builtins for custom decompilation
- More portability (OS / architectures)
- More integration between the components (tracer / debugger mostly)



# *Questions ?*

- Thank you for your attention
- If you are interested in joining us, come to talk after the conference.
- The source code of the current version (0.77b3) is available at our web CVS:
  - <http://elfsh-cvs.asgardlabs.org/>

