IDA Pro Plugins for CodeHawk-Binary

October 5, 2023

Contents

1	Introduction	2
2	Installation	2
	2.1 CodeHawk	2
	2.2 CodeHawk-Binary	3
	2.3 IDA-Pro	3
3	Analysis Preparation	3
4	IDA Pro Plugins	4
	4.1 Overview	4
	4.2 Getting Started	5
	4.3 Adding Function Entry Points	5
	4.4 Listing Function Calls	6
	4.5 Annotating Functions	8

1 Introduction

The CodeHawk-Binary analysis is a powerful analysis tool for binaries of various architectures. Its core analysis is based on the mathematical theory of abstract interpretation [3, 4]. It includes its own disassemblers, currently supporting x86, arm32 (including Thumb-2), mips, and Power32 (in progress). It takes as input a (possibly stripped) binary, disassembles it, and constructs functions, control flow graphs, and a callgraph. It then creates an over-approximating semantic translation of the functions into CHIF, the CodeHawk Internal Form, for analysis, and translates the resulting (over-approximating) invariants back into the context of the assembly code. This invariant-generation process is performed in a series of rounds, incrementally adding variables discovered in previous rounds. The final set of invariants is saved (in highly compressed form) and forms the basis for all subsequent analyses and liftings.

The CodeHawk-Binary Analyzer comes with a comprehensive command-line interpreter to produce a wide variety of analysis results, including annotated assembly code, annotated control-flow graphs and callgraphs, and potential vulnerability reports. It does not, however, have a graphical user interface. For general reverse engineering tasks a graphical user interface, such as provided by IDA Pro, Binary Ninja, Ghidra, or angr, is often a preferred way of interaction for exploration and navigation. None of these tools, however, provide the deep analysis results that CodeHawk generates. Part of the reason for a more shallow analysis in these tools is, of course, that deeper analysis often takes too much time creating response times that are not acceptable for an interactive tool.

With CodeHawk-Binary Analyzer plugins for these tools we hope to offer users the best of both worlds. CodeHawk analysis can be performed off-line. All analysis results are saved in full, such that they can be used for many different purposes, via a comprehensive python API that is called directly from the python plugin code. During an interactive session with the preferred tool, these analysis results can be accessed and deployed as desired, controlled by interactively invoking plugins, thus injecting analysis results directly into the tool's database and thereby augmenting the tool's own analysis results. Because the plugins only need to extract data, they do not perform any expensive analysis, response times are comparable to other actions typically performed in these tools.

In this report we describe and illustrate an initial selection of python plugins implemented for IDA-Pro.

2 Installation

2.1 CodeHawk

The CodeHawk core analysis is performed by codehawk, an open-source tool available from GitHub [2], implemented in ocaml. Installation instructions for codehawk are provide for Linux and MacOS in the README.

Alternatively Aarno Labs makes available a docker container that provides an easier interface to running codehawk without the need to perform any of the installation steps??

2.2 CodeHawk-Binary

The interaction with the ocaml codehawk analyzer is managed via CodeHawk-Binary, implemented in python, and also available from GitHub [1]. CodeHawk-Binary requires python to be installed with version 3.8 or higher, but does not have other dependencies, except if graphical output is requested directly from the CodeHawk-Binary command-line processor (e.g., control flow graphs or callgraphs), which requires the dot utility to be installed.

2.3 IDA-Pro

Running IDA-Pro python scripts requires a paid license for IDA-Pro. None of the IDA-Pro decompilers (sold separately by Hex-Rays) is required to run the scripts here described, so a minimal IDA-Pro license suffices.

The plugins require IDA-Pro to have CodeHawk-Binary included in its PYTHONPATH. This can be accomplished in a few different ways:

- Setting the PYTHONPATH in the terminal, if IDA-Pro is started from the command-line;
- Copying the full chb directory from the CodeHawk-Binary repository to the idabin/plugins directory;
- Making a symbolic link from the idabin/plugins directory to the chb directory (??)

The plugins themselves can simply be copied into the idabin/plugins directory from where they will be automatically loaded and initialized when IDA-Pro is started.

3 Analysis Preparation

All scripts described in the next section assume that the target binary has already been analyzed by CodeHawk.

After installation and setting of the appropriate PATH and PYTHONPATH environment variables, analysis is accomplished by the command

```
> chkx analyze <filename-of-binary> --reset
```

Depending on the size and complexity of the binary this analysis can take anywhere from a few minutes to many hours, so should typically be done off-line. It only needs to be done once¹

The analysis results will be saved in a subdirectory of the directory that holds the binary, with the name <filename-of-binary>.ch, from where they will be directly accessed by the IDA-Pro plugins.

¹In some cases we may want to feed back data from IDA-Pro into the analysis by means of analysis hints, in which case the analys must be rerun. This situation is somewhat unusual, and is described later.

4 IDA Pro Plugins

4.1 Overview

We present the first four IDA Pro plugins using CodeHawk analysis results that have been developed so far, indicated in the menu shown in Figure 1. All of them will be illustrated in more detail in the sections below.

ida64 File	Edit Jump	Search	View	Debu	gger	Lum	ina	Opt	ons	W	indow
	🗅 Сору		^C					IDA	- iot_	exa	mple
i 🤧 🔒 i -	Begin selecti	on	ΝL	1	A.			rank	, and the second se	6	.,
	Select all						Ŭ				
	Select identit	fier	企下								
Library f	Undo		ЖZ	Instruct	ion	Dat	a	Une	explor	ed	E
F Functions	Redo		ΰжΖ	× I	I	DA Viev	w-A		×	O	1
Function name	Export data		ΦE			LOA	D:	0040)53	50	
J cind_rtp	the s					LOA		0040	1530	50	#
f cmd_email	Code		C			LOA	D:	004)536	50	
f cmd block	📷 Data		D			LOA	D:	0040	0536	50	#
f cmd_netw	🚰 Struct va	r	ΣQ			LOA	D:	0040)536	50	
f cmd firew	🗱 Strings		>			LOA	D:	0040)536	50	mai
f cmd ipma	📌 Array		*			LOA	D:	0040)536	50	
angu bmo						LOA	D:	0040)536	50	
f cmd_upnp			N			LOA	D:	0040)536	50	var
F cmd_mulp	Mename		IN			LOA	D:	0040)53(50	var
F cmd_mul_	Operand type	e	>			LOA	D:	0040)536	50	var
📝 cmd_wan	Commente	-				LOA	D:	004)53	50	var
📝 cmd_log	Comments					LOA	D:	004)53	50	var
📝 cmd_sche	Segments		>			LOA	D:	004)53	50	var
📝 cmd_email	Structs		>			LOA	D:	0040)530	50	var
📝 cmd_bridg	Functions		>			LOA	D :	0040	1530	00	var
📝 cmd_wifi_r	Patch progra	m	>			LOP	D :	0040	1530	50	var
<pre>f cmd_wifi_(</pre>	Other		>			LOP	D :	0040	1530	50	var
f cmd_wifi_	Diuging			Quiek		luging		unai			
f cmd_relea	Plugins			QUICK	run p	lugins					
f cmd_greer	Start Dictatio	on		Swift							
f cmd_qos	Emoji & Syml	bols ^#	Space	SVD fi	le ma	nagem	ent			~	ÈE11
f cmd_ramov	a history			lumn	to nov	et fivun					
f cmd remov	e_nistory			One sta		(t fixup	, 		_		
F cmd emule	service ip a	dd	- 61	Create	PAT	ironi u	ie da	atabas	e		_
F cmd_apmod	de			CNKX:II	ist-ca	II-argu	men	ts			
f cmd_emule	_conn_status			chkx:a	nnota	te-fun	ctior	าร		7	ΞK
📝 cmd_rewins	5			chkx:a	nnota	ate-fun	ctior	n		7	ΓK
📝 sub_412FD	C			chkx:a	dd_fu	unction	n_ent	trypoi	nts	7	E
Line 141 of 122	4			Chang	e the	callee	addi	ress		7	F11
Line 141 01 133	+		_	Load D	OWAR	F file					5
Output				Find fu	unctio	ns					
o. creating	a new segme	ent (004D	9040-00	409894)		0	ĸ		_	_	
Flushing buff chkx:add_func chkx:annotate chkx:annotate chkx:list-cal	ers, please tion_entrypo -function in -functions i l-arguments	waitok bints init hitialized initialize initialize	iālized d ed	- <u>)</u> /	examp	ole_2/	iot_	examp	le' i	has	been

Figure 1: IDA Pro Edit menu with CodeHawk plugins (with chkx prefix)

- chkx_list_call_arguments: List all calls to a particular (user-selected) function together with their arguments;
- chkx_annotate_function: Annotate the instructions of the function at the current location of the cursor in the IDA-View;
- chkx_annotate_functions: Annotate the instructions of all functions;
- chkx_add_function_entrypoints: Add function entry points found by CodeHawk but not by IDA Pro.

4.2 Getting Started

The plugins described below are illustrated on a cgi-handling program for an http-server found on a commodity router. The binary is a dynamically linked MIPS binary (632 KB) with more than 1000 functions. The binary can be analyzed with CodeHawk with the command:

> chkx analyze iot_example

The analysis takes about 30 minutes. When finished load the binary into IDA-Pro, and invoke the plugins as described.

4.3 Adding Function Entry Points

In some cases CodeHawk analysis identifies more function entry points than IDA Pro. The plugin chkx:add_function_entrypoints facilitates adding these extra function entry points to IDA Pro automatically.

Figure 2 shows the initial IDA Pro display just after loading the example binary. Notice in particular the bar at the top, showing a colored map of of the code address range, with blue denoting the presence of functions and brown denoting code that is not part of any function. In addition, notice the number of functions found shown in bottom left of the figure: 1005.



Figure 2: IDA View after initial IDA analysis

Figure 1 shows how to invoke the plugin chkx:add_function_entrypoints: Edit \rightarrow Plugins \rightarrow chk:add_function_entrypoints. The plugin checks for every function entry point in the Code-

Hawk analysis results whether the address is contained in an IDA-Pro-recognized function; if not, it adds a new function entry point at that address and leaves the remaining function construction to IDA-Pro. The plugin takes a few seconds.

Figure 3 shows the same IDA Pro display as shown in Figure 2 after the plugin finished. Notice the effect on the code bar at the top: it now is almost entirely blue, and the number of functions, as shown in the bottom left, now stands at 1363; 358 new function entry points were added by the plugin.



Figure 3: IDA View after invoking the chkx:add_function_entrypoints plugin

Note The displays referenced above were obtained with IDA-Pro version 7.6, which was not very good at recognizing MIPS functions by preamble. The latest IDA-Pro version, version 8.3 does a much better job, and the number of functions added (with the same analysis results) is reduced to 25, as shown in Figure 4. Subsequent figures show IDA-Pro version 8.3. However, the plugins should work on any version 7.x and higher.

4.4 Listing Function Calls

IDA-Pro provides a menu option to list all calls to a particular function. Figure 5 shows the resulting display for strcpy. For every call it shows just the address of the call and the function in which the call is located. Clicking on the call moves the user to the IDA assembly view, where the call and its surroundings can be further inspected.



Figure 4: Adding function entry points to IDA-Pro, version 8.3

With CodeHawk analysis results more information can be associated with the calls, in particular, the arguments with which the function is called. Figure 6 shows result of invoking the chkx_list_call_arguments plugin (after entering the text strcpy in the presented dialog box).

The results are displayed in a standard IDA-Pro display in a new tab. As the roles of the arguments of **strcpy** are known (destination and source), the arguments can be characterized in terms of where they write to (e.g., stack or heap etc) and where the source is read from (e.g., a function return value or argument to the caller function).

The utility of having these extra columns (writes-to, reads-from) is that they can be sorted by clicking on the header of the column, like any other column, thus enabling grouping them by kind, or they can be filtered.

Figure 7 shows the result of clicking on the the **src** header, displaying all calls together that receive their input from the **cgi_value** function (a function that typically returns data that originates directly from an HTTP input form, and thus is especially interesting for vulnerability/exploitation research). Note from Figure 6 that this binary has 266 calls to **strcpy**. Only 33 of these receive input from **cgi_value**. Being able to select these (or group them) with a single click for further inspection, rather than having to inspect all 266, can considerably speed up the search for exploitable vulnerabilities.

Another function of interest for vulnerability research is, of course, sprintf. As shown in Figure 8, our example binary has 359 calls to sprintf.

Not all of these calls are interesting. In particular calls without any %s format specifiers are unlikely to be exploitable, because most other format specifiers result in a formatted argument with a fixed maximum length. Figure 9 shows how we can use the standard IDA-Pro Modify filters ... dialog to select only those calls that have a %s format specifier in the format string and then highlight those in which rtn_cgi appear in one of the varargs as the source of one of those %s specifiers. Note that none of these selections or highlightings would be possible without the function arguments supplied by the CodeHawk analysis results.

Caller 8 main 8 main 9 sub_408 sub_408 sub_408 sub_409 sub_409 sub_408 sub_408 cgi_exec sub_408 cgi_exec sub_408 sub_404 sub_408	BA4 BA4 BA4 FF4 208 60C 60C tttp_reques B70 92C	alls: strc jalr jalr jalr jalr jalr jalr jalr jalr	py store	× trcpy trcp	Hex View-
Caller 3 main 8 main 9 main 9 sub_408 8 sub_408 8 sub_408 8 sub_409 8 sub_409 8 sub_409 9 sub_409 1 sub_408 cgi_exec 8 sub_408 8 sub_408 1	BA4 BA4 FF4 2008 6DC 6DC 6DC 6DC 6DC 870 5 B70 92C	Instru jalr jalr jalr jalr jalr jalr jalr jalr	st9; s \$t9; s \$t9; s \$t9; s \$t9; st	trcpy trcpy trcpy trcpy trcpy trcpy trcpy trcpy trcpy trcpy trcpy trcpy trcpy trcpy trcpy trcpy	
B main 8 main 9 main 9 sub_408 sub_408 sub_408 sub_409 sub_409 sub_409 sub_409 sub_409 sub_409 sub_409 sub_409 sub_409 sub_409 sub_409 sub_409 sub_408 cgi_exec sub_408 sub_408 cgi_exec sub_408 sub_404 sub_408	BA4 BA4 FF4 208 6DC 6DC 6DC 100 870 870 870 870 92C	jalr jalr jalr jalr jalr jalr jalr jalr jalr jalr	\$t9; s \$t9; s \$t9; s \$t9; s \$t9; st \$t9; st \$t0; st	trcpy trcpy trcpy trcpy trcpy trcpy trcpy trcpy trcpy trcpy trcpy trcpy trcpy	
8 main main sub_408 sub_408 sub_408 sub_409 sub_409 sub_409 handle_h sub_408 cgi_exec sub_408 sub_408	BA4 BA4 FF4 208 6DC 6DC 6DC 6DC 6DC 970 870 92C	jalr jalr jalr jalr jalr jalr jalr jalr	\$t9; st \$t9; st	trcpy trcpy rcpy rcpy rcpy rcpy rcpy rcpy rcpy	
main sub_408 sub_408 sub_408 sub_409 sub_409 sub_409 handle_h sub_409 handle_h sub_408 cgi_exec sub_406 sub_406 sub_406	BA4 BA4 FF4 208 6DC 6DC 6DC 6DC 6DC 6DC 870 870 870 870 92C	jalr jalr jalr jalr jalr jalr jalr jalr	\$t9; st \$t9; st \$t0; st \$t0	trcpy rcpy rcpy rcpy rcpy rcpy rcpy rcpy	
sub_408 sub_408 sub_409 sub_409 sub_409 sub_409 handle_h sub_408 cgi_exec sub_40E sub_4106	BA4 BA4 FF4 208 6DC 6DC 6DC 870 870 870 92C	jalr jalr jalr jalr jalr jalr jalr jalr	\$t9; st \$t9; st	rcpy rcpy rcpy rcpy rcpy rcpy rcpy rcpy	
sub_408 sub_408 sub_409 sub_409 sub_409 handle_h sub_408 cgi_exec sub_40E sub_4108	BA4 FF4 208 6DC 6DC 6DC 870 870 870 92C	jalr jalr jalr jalr jalr t jalr jalr jalr	\$t9; st \$t9; st \$t9; st \$t9; st \$t9; st \$t9; st \$t9; st \$t9; st \$t9; st \$t9; st	rcpy rcpy rcpy rcpy rcpy rcpy rcpy rcpy	
sub_408 sub_409 sub_409 sub_409 handle_h sub_408 cgi_exec sub_40E sub_4108	FF4 208 6DC 6DC http_reques B70 5 B70 92C	jalr jalr jalr jalr ialr jalr jalr jalr	\$t9; st \$t9; st \$t9; st \$t9; st \$t9; st \$t9; st \$t9; st \$t9; st \$t9; st	rcpy rcpy rcpy rcpy rcpy rcpy rcpy	
sub_409 sub_409 sub_409 handle_h sub_40B cgi_exec sub_40E sub_4109	208 6DC 6DC 870 870 870 920	jalr jalr jalr ialr jalr jalr jalr	\$t9; st \$t9; st \$t9; st \$t9; st \$t9; st \$t9; st \$t9; st \$t9; st	rcpy rcpy rcpy rcpy rcpy rcpy	
sub_409 sub_409 handle_h sub_40B cgi_exec sub_40E sub_4109	6DC 6DC 1870 1970 1970 1970 1920	jalr jalr t jalr jalr jalr jalr	\$t9; st \$t9; st \$t9; st \$t9; st \$t9; st \$t9; st	rcpy rcpy rcpy rcpy rcpy	
sub_409 handle_h sub_40B cgi_exec sub_40E sub_40E	6DC http_reques B70 B70 B70 92C	jalr t jalr jalr jalr jalr	\$t9; st \$t9; st \$t9; st \$t9; st \$t9; st	rcpy rcpy rcpy rcpy	
handle_h sub_40B cgi_exec sub_40E sub_40E	http_reques B70 B70 B70 92C	t jalr jalr jalr jalr	\$t9;st \$t9;st \$t9;st \$t9;st	rcpy rcpy rcpy	
sub_40B cgi_exec sub_40E sub_4109	B70 B70 92C	jalr jalr jalr	\$t9;st \$t9;st \$t9:st	rcpy	
cgi_exec sub_40E sub_4109	: B70 92C	jalr jalr	\$t9;st \$t9:st	rcpy	
sub_40E sub_4109	B70 92C	jalr	\$t9 : st	ropy	
sub_4109	92C		+	icpy	
		jalr	\$t9;st	rcpy	
sub_4108	E34	jalr	\$t9 ; st	rcpy	
browser_	language	jalr	\$t9;st	rcpy	
do_auth		jalr	\$t9 ; st	rcpy	
is_multi_	login	jalr	\$t9 ; st	rcpy	
is_multi_	login	jalr	\$t9 ; st	rcpy	
cmd_mlc	cmd	jalr	\$t9;st	rcpy	
cmd_mlc	cmd	jalr	\$t9 ; st	rcpy	
cmd_mlc	cmd	jalr	\$t9 ; st	rcpy	
cmd_mlc	cmd	jalr	\$t9 ; st	rcpy	
cmd_mlc	cmd	jalr	\$t9;st	rcpy	
cmd_ma	caddr_rm	jalr	\$t9 ; st	rcpy	
cmd ma	caddr rm	ialr	\$t9 : st	rcov	
	cmd_mic cmd_mic cmd_ma cmd_ma	cmd_micmd cmd_micmd cmd_macaddr_rm cmd_macaddr_rm	cmd_mlcmd jah cmd_mlcmd jah cmd_mlcmd jah cmd_macaddr_rm jah cmd_macaddr_rm jah	cmd_mlcmd jalr \$19;st cmd_mlcmd jalr \$t9;st cmd_mlcmd jalr \$t9;st cmd_macaddr_rm jalr \$t9;st cmd_macaddr_rm jalr \$t9;st	cmd_mlcmd jalr \$19; strcpy cmd_mlcmd jalr \$19; strcpy cmd_macaddr_rm jalr \$19; strcpy cmd_macaddr_rm jalr \$19; strcpy

Figure 5: Native IDA-Pro display to show calls to a particular function

4.5 Annotating Functions

Similar to the native IDA-Pro display listing function calls, the displays resulting from invoking the chkx_list_call_arguments also allow navigating to the location of a call in the corresponding IDA assembly view, by double-clicking on the call in the display. For example, if we double-click on the strcpy call shown in Figure 10, the view is moved to the corresponding function in the IDA assembly view, shown in Figure 11

This display doesn't provide a lot of information beyond the bare assembly code. By invoking the chkx_annotate_function plugin while the cursor is on this function, we can make the display more informative: the plugin annotates all instructions in the function with CodeHawk analysis results. Figure 12 shows the resulting display. The display allows making a direct connection between the cgi_value key associated with the strcpy call that copies its value.

The chkx_annotate_functions plugin adds annotations to *all* functions. This may take some 10-20 secs. During its execution, progress is indicated as shown in Figure 13.

References

- [1] CodeHawk Binary Analyzer. https://github.com/static-analysis-engineering/CodeHawk-Binary.
- [2] CodeHawk Tool Suite Analyzer. https://github.com/static-analysis-engineering/codehawk.

Instruction	Data 📕 Unexplore	ed 📃 Extern	nal symbol 📕	Lumina functi	on					
× 📘 IDA	View-A 🛛 🗙 💱	strcpy calls	× 🖸	Hex View-1	\times	A	Structures	🗙 🔢 — Е	Inums	×
address	caller	callee	dst	v	vrites-to		src		1	reads-from
0x405518	0x405360	strcpy	(\$sp_in - 0x18	34) s	stack		rtn_get_macro_0>	405504		fn retval
0x4058c8	0x405360	strcpy	0x4d95a0	ç	global		rtn_config_get_0	4058b4		fn retval
0x4058f0	0x405360	strcpy	0x4d95e0	ç	global		rtn_config_get_0	4058dc		fn retval
0x408cac	0x408ba4	strcpy	(\$sp_in - 0x20	D) s	stack		rtn_config_get_0	408c94		fn retval
0x408cf8	0x408ba4	strcpy	0x4d9580	ç	global		rtn_cat_file_0x40	8cd8	·	fn retval
0x40903c	0x408ff4	strcpy	(\$sp_in - 0x13	30) s	stack		\$a0_in			fn arg
0x409238	0x409208	strcpy	(\$sp_in - 0x9	D) s	stack		\$a0_in			fn arg
0x409720	0x4096dc	strcpy	(\$sp_in - 0x2	14) s	stack		rtn_0x409708_ge	tenv("REMOTI	E	fn retval
0x4097a0	0x4096dc	strcpy	(\$sp_in - 0x2	18) s	stack		rtn_cat_file_0x40	9788		fn retval
0x40a4d0	0x409dd0	strcpy	0x4d8384	ç	global		(rtn_strstr_0x40a	4a4 + 0xd)		fn retval
0x40bc50	0x40bca8	strcpy	(\$sp_in - 0x9	0) s	stack		rtn_lang_filename	e_0x40bbc4		fn retval
0x40c1c0	0x40bd28	strcpy	0x4d9580	ç	global		rtn_config_get_0	k40c1ac		fn retval
0x40ebc4	0x40eb70	strcpy	(\$sp_in - 0x2	B) s	stack		rtn_cat_file_0x40	ebac		fn retval
0x410974	0x41092c	strcpy	(\$sp_in - 0x4a	a4) s	stack		\$a0_in			fn arg
0x410fd8	0x410e34	strcpy	(\$sp_in - 0x18	30) s	stack		rtn_config_get_0	<410fc4		fn retval
0x4116c4	0x411540	strcpy	0x4b2d10	ç	global		\$a1			unknown
0x411dec	0x411cd0	strcpy	((\$sp_in + rtn	_sprintf_0 s	stack		rtn_config_get_0	(411dd0		fn retval
0x412428	0x41238c	strcpy	(\$sp_in - 0xb	D) s	stack		rtn_cat_file_0x412	2410		fn retval
0x412450	0x41238c	strcpy	(\$sp_in - 0x70	D) s	stack		rtn_cat_file_0x412	243c		fn retval
0x4133a0	0x4132fc	strcpy	(\$sp_in - 0x9	60) s	stack		rtn_config_get_0	(413388		fn retval
0x4133cc	0x4132fc	strcpy	(\$sp_in - 0x9	20) s	stack		rtn_config_get_0	(4133b4		fn retval
0x413818	0x4132fc	strcpy	(\$sp_in - 0x4:	20) s	stack		rtn_config_get_0	(413800		ín retval
0x413840	0x4132fc	strcpy	(\$sp_in - 0x9	e0) s	stack		rtn_config_get_0	(41382c		ín retval
0x413978	0x4132fc	strcpy	(\$sp_in - 0x9a	a0) s	stack		rtn_config_get_0	(413960	·	fn retval
0x413b90	0x413af8	strcpy	(\$sp_in - 0x2′	18) s	stack		rtn_strchr_0x413b	o7c		ín retval
0x413bbc	0x413af8	strcpy	(\$sp_in - 0x2	58) s	stack		rtn_config_get_0	(413ba4		ín retval
0x4141fc	0x414170	strcpy	(\$sp_in - 0x4)	2c) s	stack		\$a0_in			fn arg
0x414248	0x414170	strcpy	(\$sp_in - 0x4	4c) s	stack		\$a1			unknown
0x414848	0x414778	strcpy	(\$sp_in - 0x11	0) s	stack		rtn_config_get_0	(414830		ín retval
0x4148a0	0x414778	strcpy	(\$sp_in - 0x11	0) s	stack		rtn_config_get_0	(41488c		ín retval
0x4148f8	0x414778	strcpy	(\$sp_in - 0x11	0) s	stack		rtn_config_get_0	(4148e4		ín retval
0x415f68	0x415f00	strcpy	rtn_malloc_0>	(415f48 h	пеар		(\$sp_in - 0x18)			stack
0x4170e4	0x417080	strcpy	(\$sp_in - 0x9	0) s	stack		rtn_get_board_int	fo_0x4170d0	·	in retval
Line 1 of 266	DV 4171E A	-	Illian in Ovol		tool/		Coll in			
2										

Figure 6: Result of invoking the the chkx_list_call_arguments plugin for strcpy

- [3] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In Robert M. Graham, Michael A. Harrison, and Ravi Sethi, editors, *Conference Record of the Fourth ACM Symposium* on Principles of Programming Languages, Los Angeles, California, USA, January 1977, pages 238–252. ACM, 1977.
- [4] Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In Alfred V. Aho, Stephen N. Zilles, and Thomas G. Szymanski, editors, Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages, Tucson, Arizona, USA, January 1978, pages 84–96. ACM Press, 1978.

Instruction	Data 📃 Unexp	lored Exter	rnal symbol 📕 Lumina	function		
🗙 📘 IDA Vi	ew-A 🛛 🗙 💱	sprintf calls	🗙 🦉 strcpy calls	🛛 🖸 🕹 🕹	View-1 × A Structures	🗙 🔚 Enums
address	caller	callee	dst	writes-to	src	reads-from
0x43aba4	0x43aa34	strcpy	(\$sp_in - 0x80)	stack	rtn_cgi_value_0x43ab88	fn retval
0x43abd8	0x43aa34	strcpy	(\$sp_in - 0x60)	stack	rtn_cgi_value_0x43abc0	fn retval
0x43ac0c	0x43aa34	strcpy	(\$sp_in - 0x40)	stack	rtn_cgi_value_0x43abf4	fn retval
0x43ad48	0x43aa34	strcpy	(\$sp_in - 0x80)	stack	rtn_cgi_value_0x43ad2c	fn retval
0x43ad7c	0x43aa34	strcpy	(\$sp_in - 0x60)	stack	rtn_cgi_value_0x43ad64	fn retval
0x43adb0	0x43aa34	strcpy	(\$sp_in - 0x40)	stack	rtn_cgi_value_0x43ad98	fn retval
0x43af40	0x43aa34	strcpy	(\$sp_in - 0x80)	stack	rtn_cgi_value_0x43af24	fn retval
0x43af74	0x43aa34	strcpy	(\$sp_in - 0x60)	stack	rtn_cgi_value_0x43af5c	fn retval
0x43afa8	0x43aa34	strcpy	(\$sp_in - 0x40)	stack	rtn_cgi_value_0x43af90	fn retval
0x43b178	0x43aa34	strcpy	(\$sp_in - 0x80)	stack	rtn_cgi_value_0x43b15c	fn retval
0x43b1ac	0x43aa34	strcpy	(\$sp_in - 0x60)	stack	rtn_cgi_value_0x43b194	fn retval
0x43b1e0	0x43aa34	strcpy	(\$sp_in - 0x40)	stack	rtn_cgi_value_0x43b1c8	fn retval
0x4402dc	0x440214	strcpy	(\$sp_in - 0x88)	stack	rtn_cgi_value_0x4402c0	fn retval
0x440310	0x440214	strcpy	(\$sp_in - 0x68)	stack	rtn_cgi_value_0x4402f8	fn retval
0x440344	0x440214	strcpy	(\$sp_in - 0x48)	stack	rtn_cgi_value_0x44032c	fn retval
0x440748	0x440688	strcpy	(\$sp_in - 0x80)	stack	rtn_cgi_value_0x44072c	fn retval
0x44077c	0x440688	strcpy	(\$sp_in - 0x60)	stack	rtn_cgi_value_0x440764	fn retval
0x4407b0	0x440688	strcpy	(\$sp_in - 0x40)	stack	rtn_cgi_value_0x440798	fn retval
0x440c48	0x440b88	strcpy	(\$sp_in - 0x80)	stack	rtn_cgi_value_0x440c2c	fn retval
0x440c7c	0x440b88	strcpy	(\$sp_in - 0x60)	stack	rtn_cgi_value_0x440c64	fn retval
0x440cb0	0x440b88	strcpy	(\$sp_in - 0x40)	stack	rtn_cgi_value_0x440c98	fn retval
0x441158	0x441098	strcpy	(\$sp_in - 0x80)	stack	rtn_cgi_value_0x44113c	fn retval
0x44118c	0x441098	strcpy	(\$sp_in - 0x60)	stack	rtn_cgi_value_0x441174	fn retval
0x4411c0	0x441098	strcpy	(\$sp_in - 0x40)	stack	rtn_cgi_value_0x4411a8	fn retval
0x444c60	0x444988	strcpy	(\$sp_in - 0x244)	stack	rtn_cgi_value_0x444a9c	fn retval
0x444fa4	0x444cfc	strcpy	(\$sp_in - 0x244)	stack	rtn_cgi_value_0x444de4	fn retval
0x447890	0x44779c	strcpy	(\$sp_in - 0x30)	stack	rtn_cgi_value_0x4477cc	fn retval
0x452278	0x452080	strcpy	0x4d95a0	global	rtn_cgi_value_0x4520bc	fn retval
0x45f97c	0x45f8e0	strcpy	(\$sp_in - 0x110)	stack	rtn_cgi_value_0x45f960	fn retval
0x45fa50	0x45f9b4	strcpy	(\$sp_in - 0x110)	stack	rtn_cgi_value_0x45fa34	fn retval
0x45fad0	0x45fa88	strcpy	(\$sp_in - 0x108)	stack	rtn_cgi_value_0x45fab4	fn retval
0x45fb4c	0x45fb04	strcpy	(\$sp_in - 0x108)	stack	rtn_cgi_value_0x45fb30	fn retval
0x462bbc	0x462b48	strcpy	(\$sp_in - 0x2c4)	stack	rtn_cgi_value_0x462ba0	fn retval

Figure 7: Sorting the result of invoking the chkx_list_call_arguments plugin for strcpy

Instruction	Data	Unexplored	External symbol	Lumina function	on	
🗙 📑 IDA	View-A	🗙 🥎 sprintf d	calls 🛛 🗙 🕎 s	trcpy calls \times	🖸 Hex View-1 🛛 🛪 🖪 Structures 🛛 🗶 ☷	Enums 🛛 🛛 📷 Imports 🛛 🔀 Exports
address	caller	callee	dst	writes-to	fmt	varargs
0x405e8c	0x405e18	sprintf	(\$sp_in - 0x98)	stack	"smartctl -x /dev/%s > %s"	rtn_config_get_0x405e68, "/tmp/configs-backup.lrp"
0x406108	0x405f48	sprintf	\$a0_in	fn arg	"%s %s %s %s %s%s.\n"	rtn_get_lang_0x40608c, \$a3, var.0336, var.0332, 0x4b
0x408b28	0x408a0c	sprintf	(\$sp_in - 0x90)	stack	"/sbin/mtd write %s /dev/mtd/0 > /dev/null"	"/tmp/uboot-upgrade.bin"
0x4099e0	0x4098dc	sprintf	(\$sp_in - 0x98)	stack	"%d %s"	\$a1_in, \$a2_in
0x40a308	0x409dd0	sprintf	(\$sp_in - 0x58)	stack	"access_control%d"	\$a2
0x40d8fc	0x40d890	sprintf	(\$sp_in - 0xa8)	stack	"%s%d"	\$a0_in, \$a3
0x40d91c	0x40d890	sprintf	(\$sp_in - 0x68)	stack	"%s%d"	\$a0_in, \$a3
0x40d9f0	0x40d998	sprintf	(\$sp_in - 0x58)	stack	"%s%d"	\$a0_in, \$a3
0x40dae8	0x40da4c	sprintf	(\$sp_in - 0x2e8)) stack	"%s%d"	\$a0_in, \$a3
0x40db04	0x40da4c	sprintf	(\$sp_in - 0x2a8)) stack	"%s%d"	\$a0_in, \$a3
0x40db28	0x40da4c	sprintf	(\$sp_in - 0x268) stack	"%s%d"	\$a0_in, \$a3
0x40db50	0x40da4c	sprintf	(\$sp_in - 0x228) stack	"%s"	rtn_config_get_0x40db38
0x40dcb0	0x40dbf0	sprintf	(\$sp_in - 0x2a8)) stack	"%s%d"	\$a0_in, \$a3
0x40dcd0	0x40dbf0	sprintf	(\$sp_in - 0x268) stack	"%s%d"	\$a0_in, \$a3
0x40dcf8	0x40dbf0	sprintf	(\$sp_in - 0x228)) stack	"%s"	rtn_config_get_0x40dce0
0x40dd84	0x40dd2c	sprintf	(\$sp_in - 0x58)	stack	"%s%d"	\$a0_in, \$a3
0x411db8	0x411cd0	sprintf	(\$sp_in - 0x90)	stack	"%s:"	rtn_config_get_0x411d74
0x411ed4	0x411e50	sprintf	(\$sp_in - 0x88)	stack	"%ld"	\$a2
0x412130	0x411fcc	sprintf	(\$sp_in - 0xa0)	stack	"%ld"	\$a2
0x412ba0	0x412b54	sprintf	(\$sp_in - 0xd8)	stack	"cat %s sed '/^%s/d' > /tmp/dl_downloaded"	\$a0_in, \$a3
0x412bcc	0x412b54	sprintf	(\$sp_in - 0xd8)	stack	"cat %s sed '/^%s/d' > /tmp/greendownload/work	\$a0_in, \$a3
0x412c54	0x412bfc	sprintf	(\$sp_in - 0x50)	stack	"dlclient -c serverdel -l '%s:%s' >/dev/console"	\$a0_in, \$a1_in
0x412ce0	0x412c88	sprintf	(\$sp_in - 0x90)	stack	"dlclient -c serveradd -l '%s:%s' >/dev/console"	\$a0_in, \$a1_in
0x412d6c	0x412d14	sprintf	(\$sp_in - 0x88)	stack	"ping -c 4 %s > %s &"	"www.netgear.com", "/tmp/ping_result"
0x412e90	0x412dac	sprintf	(\$sp_in - 0x98)	stack	"dlclient -c connect -l '%s:%s' >/dev/console"	\$a1_in, \$a2_in
0x413700	0x4132fc	sprintf	(\$sp_in - 0x820) stack	"dlclient -c priority -i %s -n %s >/dev/console"	(\$sp_in - 0x9e8), rtn_config_get_0x4136dc
0x4137a4	0x4132fc	sprintf	(\$sp_in - 0x820) stack	"dlclient -c %s -L /tmp/result"	(\$sp_in - 0x9f0)
0x4138a8	0x4132fc	sprintf	(\$sp_in - 0x820) stack	"dlclient -c add -t ftp -l '%s' -u %s -p %s -s %s -L /	(\$sp_in - 0x420), (\$sp_in - 0x9e0), 0x4b1bf0, var.2548,
0x4138f8	0x4132fc	sprintf	(\$sp_in - 0x820) stack	"dlclient -c %s -n '%s' -L /tmp/result"	(\$sp_in - 0x9f0), (\$sp_in - 0x8a0)
0x413f38	0x413e78	sprintf	(\$sp_in - 0x2a8)) stack	"%s%d"	\$a0_in, \$a3
0x413f58	0x413e78	sprintf	(\$sp_in - 0x268) stack	"%s%d"	\$a0_in, \$a3
0x413f80	0x413e78	sprintf	(\$sp_in - 0x228)) stack	"%s"	rtn_config_get_0x413f68
0x414574	0x414510	sprintf	(\$sp_in - 0x28)	stack	"%d"	rtn_enable_ap_0x41453c
0.41450	0.414510	anrintf	(then in Oute)	at a al /	וול זעו	rtn anabla an Ov111EE1
Line 1 of 359	,					

Figure 8: Result of invoking the chkx_list_call_arguments plugin for sprintf

Library function 📃 Regular function 📕	Instruction 📃 Data 📕 Unexplored 📒	External symbol 📃 Lumina function		
unctions 💿 💿 😒	🗙 📘 IDA View-A 🛛 🗙 💔 sprintf o	calls 🛛 🗙 📢 strcpy calls 🛛 🗙 🚺 Hex	View-1 × 🖪 Structures	🗙 🔃 Enums 🛛 🗙 🛐 Imports 🛛 🗶 🛃 Exports
• • •	Modify filters	dat uuitaa ta faat		varargs \$a2, "sectype" \$a2, "auth"
If column (any) 📀 contains	C rtn_cgi	🕑 then 🖌 highlight 🤇		\$a2, "key_length" \$a2, "key"
Match case Regular expression	Highlight	Add Reset Close		\$a2, "key" (\$sp_in - 0xac), \$a3 \$a2, "wep"
Filter list	Value	Action	%d"	(\$sp_in - 0xac), \$a3, var.0232 \$a2, "sectype"
Vara contains	Column Column Value Action X vara contains rtn_cgi highlight			"wl_", "ssid" "wl_", "hidden_channel" "ut " "beeneel"
✓ ✓ ✓ ✓ ✓ fmt contains	rtn_cgi %s	include		"w_, "cnanner" "wl_", "simple_mode" "wl_", "simple_mode"
e t				"wla_", "ssid" "wla_", "hidden_channel" "wida_" "beanse!"
E				"wla_", "simple_mode" "wl_", "simple_mode"
	UX4432/0 UX442334 Spiniu	(\$\$P_III - UXdO) SLOCK 705705		"wl_", "simple_mode" "wl_", "simple_mode"
thread_create	0x4437d0 0x4432a0 sprintf 0x443e50 0x443d78 sprintf	(\$sp_in - 0x48) stack "ifconfig (\$sp_in - 0x118) stack "%s %s"	ppp0 mtu <mark>%s</mark> "	rtn_cgi_value_0x443300 rtn_cgi_value_0x443de0, rtn_cgi_value_0x443e00
onfig_unset onfig_restore	0x44409c 0x44401c sprintf 0x444674 0x444440 sprintf	<mark>(\$sp_in - 0x110) stack "%s</mark> %s" (\$sp_in - 0x228) stack " <mark>%s</mark> %s"		rtn_cgi_value_0x444050, rtn_cgi_value_0x444070 rtn_0x444504_getenv("REMOTE_ADDR"), rtn_config_g
sonect scv	0x444a7c 0x444988 sprintf 0x444dc4 0x444cfc sprintf	(\$sp_in - 0x234) stack " <mark>%s</mark> %s ' (\$sp_in - 0x234) stack " <mark>%s</mark> %s '	%s" %s"	rtn_cgi_value_0x444a04, rtn_cgi_value_0x444a28, var rtn_cgi_value_0x444d4c, rtn_cgi_value_0x444d70, var
open trtok r	0x4451ec 0x445180 sprintf 0x445278 0x445180 sprintf	(\$sp_in - 0x268) stack "%s%d" (\$sp_in - 0x228) stack "%s%s%	s%s%s"	\$a0_in, \$a3 rtn_config_get_0x4451fc, \$a1_in, 0x4b1bf0, var.0620, v
ctl aitpid	0x4452b4 0x445180 sprintf 0x445fb4 0x445d1c sprintf	(\$sp_in - 0x228) stack "%s%s% (\$sp_in - 0x2a8) stack "%s"	S"	rtn_config_get_0x4451fc, \$a1_in, 0x4b1bf0 rtn_config_get_0x445f98 \$2. "websare at http://
rcpy utchar	0x446698 0x445d1C sprintf 0x448590 0x448520 sprintf 0x448700 0x448520 sprintf	(\$sp_in - 0x124) stack "/usr/sbin (\$sp_in - 0x224) stack "/usr/sbin	<mark>\\$</mark> /%5 [™] h/dev-scan <mark>%s</mark> > /dev/console" > <mark>%e</mark> %e%e 2 %e %e"	saz, ™weicomeok.ntm" rtn_config_get_0x448570 (\$ep.in.e.0x230) (\$ep.in.e.0x22c) 0x4b1bf0 vor 0€64
1358 of 1359	Line 128 of 206	State Pitona		

Figure 9: Filtering and highlighting the result of invoking the chkx_list_call_arguments plugin for sprintf

0x43b178	0x43aa34	strcpy	(\$sp_in - 0x80)	stack	rtn_cgi_value_0x43b15c	tn retval
0x43b1ac	0x43aa34	strcpy	(\$sp_in - 0x60)	stack	rtn_cgi_value_0x43b194	fn retval
0x43b1e0	0x43aa34	strcpy	(\$sp_in - 0x40)	stack	rtn_cgi_value_0x43b1c8	fn retval
0x4402dc	0x440214	strcpy	(\$sp_in - 0x88)	stack	rtn_cgi_value_0x4402c0	fn retval
0x440310	0x440214	strcpy	(\$sp_in - 0x68)	stack	rtn_cgi_value_0x4402f8	fn retval
0x440344	0x440214	strcpy	(\$sp_in - 0x48)	stack	rtn_cgi_value_0x44032c	fn retval
0x440748	0x440688	strcpy	(\$sp_in - 0x80)	stack	rtn_cgi_value_0x44072c	fn retval

Figure 10: Select one of the calls in the result of invoking the chkx_list_call_arguments plugin for strcpy

Instruction	Data Unexplored	External symbol	Lumina function
× 🖪 IDA	View 🗙 💱 system ca	🛛 🛛 🥎 sprintf ca	🛛 🗙 🕎 strcpy ca 🗶 🖸 Hex View 👋 🖪 Structur 👋 🗄 Enums 👋 🛐 Impo 👋 📝 Exp
•	LOAD:00440278	sw \$zer	co, 0x84+var_44(\$sp)
•	LOAD:0044027C	sw \$zer	co, 0x84+var_40(\$sp)
•	LOAD:00440280	sw \$zer	ro, 0x84+var_3C(\$sp)
•	LOAD:00440284	sw \$zer	co, 0x84+var_38(\$sp)
•	LOAD:00440288	sw \$zer	co, 0x84+var_34(\$sp)
•	LOAD:0044028C	sw \$zer	co, 0x84+var 30(\$sp)
•	LOAD:00440290	sw \$zer	ro, 0x84+var 2C(\$sp)
	LOAD:00440294	sw \$zer	co, 0x84+var_28(\$sp)
	LOAD:00440298	sw \$zer	co, 0x84+var_24(\$sp)
	LOAD:0044029C	sw \$zer	co, 0x84+var_20(\$sp)
	LOAD:004402A0	sw \$zer	:o, 0x84+var_1C(\$sp)
	LOAD:004402A4	sw \$zer	co, 0x84+var_18(\$sp)
	LOAD:004402A8	sw \$zer	co, 0x84+var_14(\$sp)
	LOAD:004402AC	sw \$zer	co, 0x84+var_10(\$sp)
	LOAD:004402B0	sw \$zer	ro, 0x84+var_C(\$sp)
	LOAD:004402B4	sw \$zer	ro, 0x84+var_8(\$sp)
	LOAD:004402B8	addiu \$a0,	(aDnsassign - 0x490000) # "DNSAssign"
	LOAD:004402BC	move \$al,	\$\$4
	LOAD:004402C0	jalr \$t9	; cgi_value
	LOAD:004402C4	move \$a2,	\$s5
	LOAD:004402C8	beqz \$v0,	loc_440360
	LOAD:004402CC	lw \$gp,	0x84+var_6C(\$sp)
1 I I I I I I I I I I I I I I I I I I I	LOAD:004402D0	la \$t9,	strcpy
1 · · · ·	LOAD:004402D4	addiu \$s2,	\$sp, 0x84+var_64
	LOAD:004402D8	move \$a1,	\$v0
	LOAD:004402DC	jalr \$t9	; strcpy
	LOAD:004402E0	move \$a0,	\$52
	LOAD:004402E4	lw \$gp,	0x84+var_6C(\$sp)
	LOAD:004402E8	lui \$a0,	0x48 # 'H'
	LOAD:004402EC	la \$t9,	cgi_value
	LOAD:004402F0	move \$a2,	\$s5
	LOAD:004402F4	li \$a0,	aEtherDnsaddrl # "ether_dnsaddrl"
1 I I I I I I I I I I I I I I I I I I I	LOAD:004402F8	jalr \$t9	; cgi_value
	LOAD:004402FC	move \$a1,	\$\$4
	LOAD:00440300	lw \$gp,	0x84+var_6C(\$sp)
	LOAD:00440304	addiu \$s1,	\$sp, 0x84+var_44
	LOAD:00440308	la \$t9,	strcpy
	LOAD:0044030C	move \$a1,	\$v0
	LOAD:00440310	jalr \$t9	; stropy
	LOAD:00440314	move \$a0,	Şsl
	LOAD:00440318	lw \$gp,	0x84+var_6C(\$sp)
I I I	LOAD:0044031C	lui \$a0,	0x48 # 'H'

Figure 11: Standard IDA-Pro assembly view for function sub_440214

		-	
Instruction	Data Upovplored	Extornal symbo	
Instruction		External symbo	
	View X 🖤 system ca	🗙 🕦 sprin	utfica
	1000-00440278		
1 .	LOAD:00440278	SW	z_{2210} , $z_{201} + 4(z_{20}) + v_{21} + 0.04 + 0.070$
	LOAD:0044027C	SW	z_{2}
•	LOAD:00440280	SW	z_{2} = 0, 0, z_{3} + z_{1} = z_{3} (z_{2}) $+ z_{2}$ (z_{2}) $+ z_{2}$ (z_{2}) $+ z_{3}$ (z_{2}) (z_{2}) $+ z_{3}$ (z_{2}) (
	LOAD:00440284	Sw SW	z_{2210} , z_{31} , $z_$
•	LOAD:00440280	ow	$\sqrt{2}$
•	LOAD:0044020C	GW	$\frac{1}{2}$
•	LOAD • 00440294	ew	$z_{2} = 0.023 + v_{1} - 22(z_{2}) + v_{1} - 0.076 = 0.00$
•	LOAD:00440298	SW	Szero, $0x84yar_{24}(5s) = 4xr_{0072} = 0x0$
•	LOAD:0044029C	sw	z_{zero} , $0x_{zero}$, $20(z_{zer})$, z_{zero} , $0x_{zero}$, $0x_$
•	LOAD:004402A0	sw	S_{2} = 0, 0.884+var 1C(Sp) # var. 0.064 := 0.0
•	LOAD:004402A4	sw	s_{zero} , $0.884 + var = 18(sp) # var, 0.060 := 0.000000000000000000000000000000000$
•	LOAD:004402A8	SW	s_{zero} , $0x84+var$ 14(s_{pp}) # var , 0056 := 0x0
•	LOAD:004402AC	SW	s_{zero} , $0x84+var 10(sp) # var, 0052 := 0x0$
•	LOAD:004402B0	SW	$s_{zero}, 0x84+var C(sp) # var,0048 := 0x0$
•	LOAD:004402B4	SW	$s_{zero}, 0x84+var 8(ssp) # var, 0044 = 0x0$
•	LOAD:004402B8	addiu	\$a0, (aDnsassign - 0,, 0,, 0,, 9,, 9,, 1768) (= "DNSAssign")
•	LOAD:004402BC	move	\$a1, \$s4 # \$a1 := \$a0 in
•	LOAD:004402C0	jalr	<pre>\$t9 ; cgi value # call App:cgi value("DNSAssign", \$0 in, \$a1 in, \$a3 in)</pre>
•	LOAD:004402C4	move	\$a2, \$s5 \$a2 := \$a1 in
	LOAD:004402C8	begz	v_0 , loc 440360 # if $v_0 = v_0$ (= (min eqr value 0x4402c0 == 0x0)) then goto 0x440360
•	LOAD:004402CC	lw	p_{gp} , $0x84+var$ 6C(p_{gp}) # $p_{gp} := 0x4b1bf0$
•	LOAD:004402D0	la	\$t9, strcpy # \$t9 := 0x46f7f0
	LOAD:004402D4	addiu	s_2 , s_p , $0x84+var$ 64 # s_2 := (s_p + $0x20$) (= (s_p n - $0x88$))
	LOAD:004402D8	move	<pre>\$a1, \$v0 # \$a1 := rtn_cgi_value_0x4402c0</pre>
	LOAD:004402DC	jalr	<pre>\$t9 ; strapy # call strcpy((\$sp_in - 0x88), rtn_cgi_value_0x4402c0)</pre>
	LOAD:004402E0	move	\$a0, \$s2 # \$a0 := (\$sp_in - 0x88)
	LOAD:004402E4	lw	<pre>\$gp, 0x84+var_6C(\$sp) # \$gp := 0x4blbf0</pre>
	LOAD:004402E8	lui	\$a0, 0x48 # 'H' # \$a0 := 0x480000
	LOAD:004402EC	la	<pre>\$t9, cgi_value # \$t9 := 0x40cdfc</pre>
	LOAD:004402F0	move	\$a2, \$s5
	LOAD:004402F4	11	<pre>\$a0, aEtherDnsaddar # \$a0 := (\$a0 + 0x72d0) (= "ether_dnsaddr1")</pre>
	LOAD:004402F8	jalr	<pre>\$t9 ; cgi_value</pre>
	LOAD:004402FC	move	\$a1, \$s4 # \$a1 = \$a0 in
	LOAD:00440300	TM	sgp , $0x84+var_6C(sp)$ # sgp := $0x4DiDi0$
	LOAD:00440304	addiu	s_{s1} , s_{sp} , u_{x4+var} , $44 \# s_{s1} := (s_{sp} + u_{x40}) (= (s_{sp} - u_{x68}))$
	LOAD:00440308	la	\$t9, stropy # \$t9 := 0x461/10
	LOAD: 0044030C	move	a_1 , a_2 , a_3 , a_1 := rtn cg1 value 0x440/rt8
	LOAD:00440310	jair	<pre>\$ty ; stropy # call stropy ((\$sp_in - uxb8), rtn_cgi_value_0x4402f8)</pre>
	LOAD:00440314	move	au, au := (au) = (au)

Figure 12: Standard IDA-Pro assembly view for function $\mathtt{sub_440214}$ annotated with CodeHawk results data



Figure 13: Progress indicator for chkx_annotate_functions