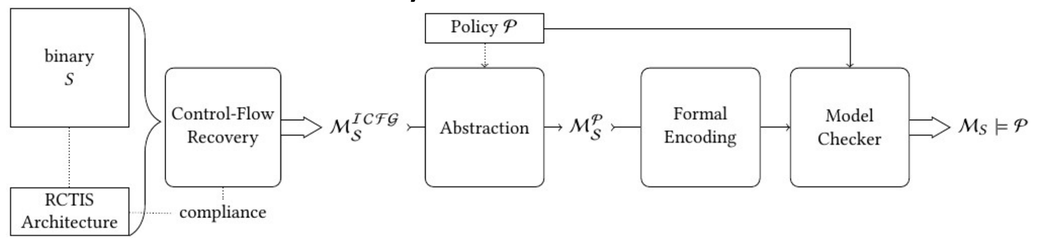
Towards Verified Security Policies for Binaries

Abstract

Approaches for the automatic assessment of security properties on source code level cannot trivially be applied to binaries. This is due to the lacking high-level semantics of low-level object code, and the fundamental problem that interprocedural control-flow recovery from a binary is difficult. We present a novel approach to statically extract abstract models of the potential behavior of a given binary and verify security policies on these models using model checking. The key ideas of our approach are twofold. First, we define a restricted control transition instruction set (RCTIS), which restricts the number of possible targets for each branch to a finite number of given targets. With that, we can, for all RCTIS compliant binaries, efficiently compute a safe overapproximation of the interprocedural control flow. Second, we propose a concept for the expression of generalized security policies by reasoning over the context under which low-level API calls are executed with temporal logics. We demonstrate the applicability of our approach by showing how confidentiality can be verified for a sample binary.

Security Verification on Binaries



The goal of our work is to develop a method for the assessment of the security of a given binary with respect to a set of different security policies. It takes as input a given binary *S* and a security policy *P* and checks whether the potential behavior M_S, models the policy in question. The behavior model is created through reconstruction of the interprocedural control-flow graph. While this does contain sufficient information, performing verification on it is ill-advised. Instead, an abstract, policy-specific model M^P should be created first. We reason that retaining information about the use of low-level APIs, such as system calls, is sufficient to express generalized security policies.

Restricted Control Transition Instruction set

Starting from object code, the reconstruction of the ICFG is by itself an open computer security issue. As control hijacking attacks show, the control-flow of binaries is often vulnerable to arbitrary control redirection. Binaries vulnerable to such attacks could potentially exhibit any possible behavior. We propose the concept of a restricted control transition instruction set, as a restriction of the control transition instructions. The modifications are based on recent findings, most notably control-flow integrity.
If a binary complies with RCTIS, it is possible to efficiently compute a safe overapproximation of its ICFG based on the information stored in the target table *TT*.

Case	Transition Type	Form	Semantics	Possible Successors
1	Fall through	*do e T*	$(u,\sigma) \vdash (\Gamma(u), \sigma[T \leftarrow [\![e]\!]\sigma])$	$\{\Gamma(u)\}$
2	Direct Branch	*br T*	$(u,\sigma) \vdash (T,\sigma)$	$\{T\}$
3	Conditional Branch	*br e T*	$(u,\sigma) \vdash \begin{cases} (T,\sigma) & ,[\![e]\!]\sigma = 0 \\ (\Gamma(u),\sigma) & ,\text{otherwise} \end{cases}$	$\{T, \Gamma(u)\}$
4	Indirect Branch	*br e*	$(u,\sigma) \vdash ([\![e]\!]\sigma, \sigma)$	V
4*	Restricted Branch	*br TT e*	$(u,\sigma) \vdash \begin{cases} ([\![e]\!]\sigma,\sigma) & ,[\![e]\!]\sigma \in TT \\ (halt,\sigma) & ,\text{otherwise} \end{cases}$	TT

Generalized Security Policies

As a consequence of the abstract nature of security policies, the concrete ICFG model, is not a suitable representation of the potential behavior for model checking. To achieve a universally applicable description of security policies, we define an abstract behavior model M^P, which contains only the information necessary to prove *P*. We propose to retain information about the context and dependencies of low-level APIs, such as system calls. The functionality offered by them is mostly standardized, well-documented and openly accessible. Furthermore, the functionality these interfaces offer are not defined by the binary itself, thus cannot be altered by the authors.

As an example, **confidentiality** holds for a binary S, if, for all paths where a read system call may be followed by a write system call, and additionally a data dependence between the contents of the buffer written and the contents of the buffer read exists, the classification level of the target file descriptor is equal or higher than the classification level of the source file descriptor. This can be expressed with CTPL logic.

$$\begin{aligned} \mathcal{P} := \forall s,t: \quad & \mathbf{AG}(n = read(s, b_r) \land \mathbf{EF}(write(t, b_w) \land \\ & \exists c \in [b_r, b_r + n], d \in [b_w, b_w + n] : c \rightarrow_d d)) \\ & \implies \mathrm{CL}(t) \geq \mathrm{CL}(s) \end{aligned}$$

Verification

To automate this verification process, we intend to use a model checker. Using a formal encoding of the abstract behavior model M^P, and the security policy as a property in temporal logic, security verification can be rephrased as a model checking problem.

$$\mathcal{M}_S \models \mathcal{P}$$

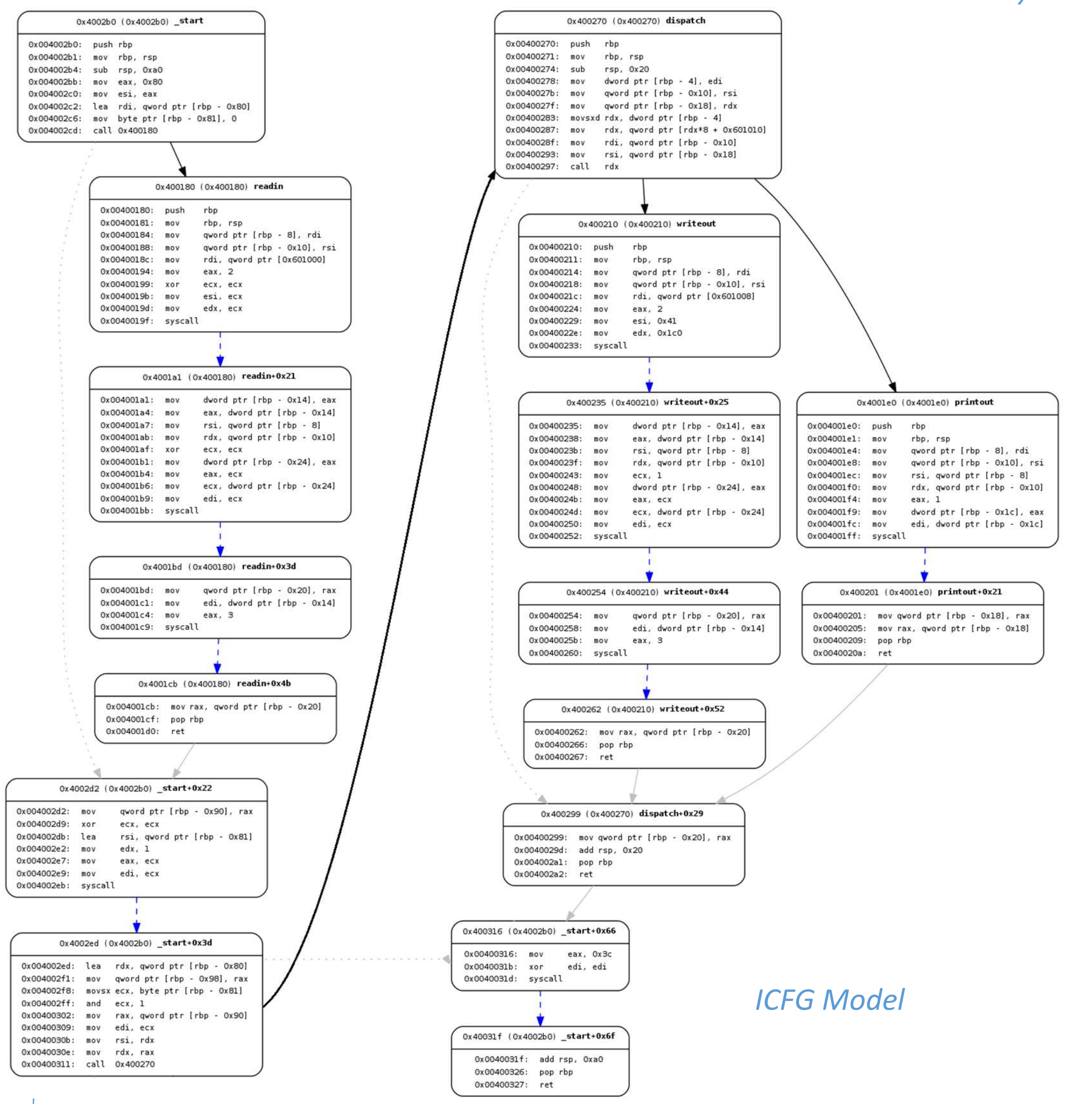
Example

```c
const char* infile  = "./file";
const char* outfile = "./copy";

ssize_t readin(char* buf, size_t count) {
    int fd;
    ssize_t result;
    _open(infile, O_RDONLY, 0, fd);
    _read(fd, buf, count, result);
    _close(fd);
    return result;
}

ssize_t printout(char* buf, size_t count) {
    ssize_t result;
    _write(1, buf, count, result);
    return result;
}

ssize_t writeout(char* buf, size_t count) {
    int fd;
    ssize_t result;
    _open(outfile, O_WRONLY|O_CREAT, S_IRWXU, fd);
    _write(fd, buf, count, result);
    _close(fd);
    return result;
}
```

```c
typedef ssize_t (*fptr)(char* buf, size_t count);
fptr cmds[] = {writeout, printout};

void dispatch(int cmd, char* buf, size_t count) {
    cmds[cmd](buf, count);
    return;
}

void _start() {
    char buf[128];
    char cmd = 0;
    ssize_t count, result;
    count = readin(buf, 128);
    _read(0, &cmd, 1, result);
    dispatch(cmd & 1, buf, count);
    _exit(0);
}
```

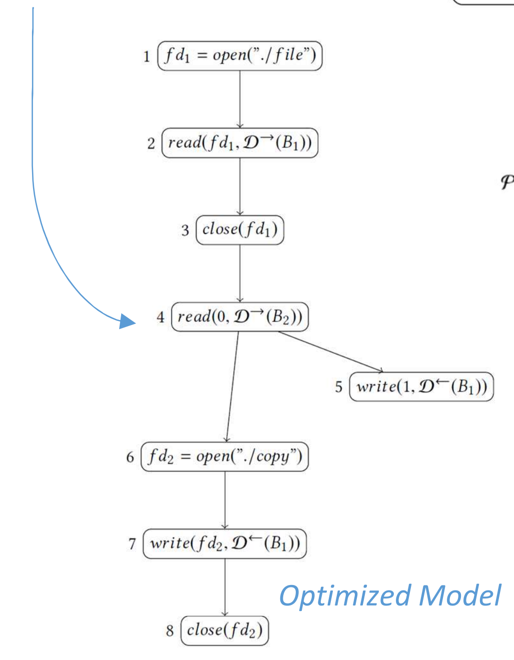
RCTIS compliant binary

Control Flow Recovery



```
0x4002b0 (0x4002b0) _start
0x004002b0:  push rbp
0x004002b1:  mov  rbp, rsp
0x004002b4:  sub  rsp, 0xa0
0x004002bb:  mov  eax, 0x80
0x004002c0:  mov  esi, eax
0x004002c2:  lea  rdi, qword ptr [rbp - 0x80]
0x004002c6:  mov  byte ptr [rbp - 0x81], 0
0x004002cd:  call 0x400180

0x400180 (0x400180) readin
0x00400180:  push   rbp
0x00400181:  mov    rbp, rsp
0x00400184:  mov    qword ptr [rbp - 8], rdi
0x00400188:  mov    qword ptr [rbp - 0x10], rsi
0x0040018c:  mov    rdi, qword ptr [0x601000]
0x00400194:  mov    eax, 2
0x00400199:  xor    ecx, ecx
0x0040019b:  mov    esi, ecx
0x0040019d:  mov    edx, ecx
0x0040019f:  syscall

0x4001a1 (0x400180) readin+0x21
0x004001a1:  mov    dword ptr [rbp - 0x14], eax
0x004001a4:  mov    eax, dword ptr [rbp - 0x14]
0x004001a7:  mov    rsi, qword ptr [rbp - 8]
0x004001ab:  mov    rdx, qword ptr [rbp - 0x10]
0x004001af:  xor    ecx, ecx
0x004001b1:  mov    dword ptr [rbp - 0x24], eax
0x004001b4:  mov    eax, ecx
0x004001b6:  mov    ecx, dword ptr [rbp - 0x24]
0x004001b9:  mov    edi, ecx
0x004001bb:  syscall

0x4001bd (0x400180) readin+0x3d
0x004001bd:  mov    qword ptr [rbp - 0x20], rax
0x004001c1:  mov    edi, dword ptr [rbp - 0x14]
0x004001c4:  mov    eax, 3
0x004001c9:  syscall

0x4001cb (0x400180) readin+0x4b
0x004001cb:  mov rax, qword ptr [rbp - 0x20]
0x004001cf:  pop rbp
0x004001d0:  ret

0x4002d2 (0x4002b0) _start+0x22
0x004002d2:  mov    qword ptr [rbp - 0x90], rax
0x004002d9:  xor    ecx, ecx
0x004002db:  lea    rsi, qword ptr [rbp - 0x81]
0x004002e2:  mov    edx, 1
0x004002e7:  mov    eax, ecx
0x004002e9:  mov    edi, ecx
0x004002eb:  syscall

0x4002ed (0x4002b0) _start+0x3d
0x004002ed:  lea   rdx, qword ptr [rbp - 0x80]
0x004002f1:  mov   qword ptr [rbp - 0x98], rax
0x004002f8:  movsx ecx, byte ptr [rbp - 0x81]
0x004002ff:  and   ecx, 1
0x00400302:  mov   rax, qword ptr [rbp - 0x90]
0x00400309:  mov   edi, ecx
0x0040030b:  mov   rsi, rdx
0x0040030e:  mov   rdx, rax
0x00400311:  call  0x400270

0x400270 (0x400270) dispatch
0x00400270:  push   rbp
0x00400271:  mov    rbp, rsp
0x00400274:  sub    rsp, 0x20
0x00400278:  mov    dword ptr [rbp - 4], edi
0x0040027b:  mov    qword ptr [rbp - 0x10], rsi
0x0040027f:  mov    qword ptr [rbp - 0x18], rdx
0x00400283:  movsxd rdx, dword ptr [rbp - 4]
0x00400287:  mov    rdx, qword ptr [rdx*8 + 0x601010]
0x0040028f:  mov    rdi, qword ptr [rbp - 0x10]
0x00400293:  mov    rsi, qword ptr [rbp - 0x18]
0x00400297:  call   rdx

0x400210 (0x400210) writeout
0x00400210:  push   rbp
0x00400211:  mov    rbp, rsp
0x00400214:  mov    qword ptr [rbp - 8], rdi
0x00400218:  mov    qword ptr [rbp - 0x10], rsi
0x0040021c:  mov    rdi, qword ptr [0x601008]
0x00400224:  mov    eax, 2
0x00400229:  mov    esi, 0x41
0x0040022e:  mov    edx, 0x1c0
0x00400233:  syscall

0x400235 (0x400210) writeout+0x25
0x00400235:  mov    dword ptr [rbp - 0x14], eax
0x00400238:  mov    eax, dword ptr [rbp - 0x14]
0x0040023b:  mov    rsi, qword ptr [rbp - 8]
0x0040023f:  mov    rdx, qword ptr [rbp - 0x10]
0x00400243:  mov    ecx, 1
0x00400248:  mov    dword ptr [rbp - 0x24], eax
0x0040024b:  mov    eax, ecx
0x0040024d:  mov    ecx, dword ptr [rbp - 0x24]
0x00400250:  mov    edi, ecx
0x00400252:  syscall

0x4001e0 (0x4001e0) printout
0x004001e0:  push   rbp
0x004001e1:  mov    rbp, rsp
0x004001e4:  mov    qword ptr [rbp - 8], rdi
0x004001e8:  mov    qword ptr [rbp - 0x10], rsi
0x004001ec:  mov    rsi, qword ptr [rbp - 8]
0x004001f0:  mov    rdx, qword ptr [rbp - 0x10]
0x004001f4:  mov    eax, 1
0x004001f9:  mov    dword ptr [rbp - 0x1c], eax
0x004001fc:  mov    edi, dword ptr [rbp - 0x1c]
0x004001ff:  syscall

0x400254 (0x400210) writeout+0x44
0x00400254:  mov    qword ptr [rbp - 0x20], rax
0x00400258:  mov    edi, dword ptr [rbp - 0x14]
0x0040025b:  mov    eax, 3
0x00400260:  syscall

0x400201 (0x4001e0) printout+0x21
0x00400201:  mov qword ptr [rbp - 0x18], rax
0x00400205:  mov rax, qword ptr [rbp - 0x18]
0x00400209:  pop rbp
0x0040020a:  ret

0x400262 (0x400210) writeout+0x52
0x00400262:  mov rax, qword ptr [rbp - 0x20]
0x00400266:  pop rbp
0x00400267:  ret

0x400299 (0x400270) dispatch+0x29
0x00400299:  mov qword ptr [rbp - 0x20], rax
0x0040029d:  add rsp, 0x20
0x004002a1:  pop rbp
0x004002a2:  ret

0x400316 (0x4002b0) _start+0x66
0x00400316:  mov    eax, 0x3c
0x0040031b:  xor    edi, edi
0x0040031d:  syscall

0x40031f (0x4002b0) _start+0x6f
0x0040031f:  add rsp, 0xa0
0x00400326:  pop rbp
0x00400327:  ret
```

ICFG Model


1 $fd_1 = open("./file")$
2 $read(fd_1, \mathcal{D}^{\rightarrow}(B_1))$
3 $close(fd_1)$
4 $read(0, \mathcal{D}^{\rightarrow}(B_2))$
5 $write(1, \mathcal{D}^{\leftarrow}(B_1))$
6 $fd_2 = open("./copy")$
7 $write(fd_2, \mathcal{D}^{\leftarrow}(B_1))$
8 $close(fd_2)$

Optimized Model

Confidentiality Policy

$$\begin{aligned} \mathcal{P} := \forall s,t: \quad & \mathbf{AG}(read(s, \mathcal{D}^{\rightarrow}(B_r)) \land \mathbf{EF}(write(t, \mathcal{D}^{\leftarrow}(B_w)) \land \\ & \mathcal{D}^{\rightarrow}(B_r) \cap \mathcal{D}^{\leftarrow}(B_w) \neq \emptyset)) \\ & \implies \mathrm{CL}(t) \geq \mathrm{CL}(s) \end{aligned}$$

Model Checker

This binary may:
- copy information from `./file` to `./copy`
- leak information from a `./file` to `stdout`

Tobias Pfeffer
Technische Universität Berlin
Software and Embedded Systems Engineering
tobias.pfeffer@tu-berlin.de