Vulnerability Flow Type Systems

Mohsen Lesani
University of California, Santa Cruz

Overview

- Introduction
- Type System
- Operational Semantics
- Type-safety Guarantees

Attacks Compose Vulnerabilities

- Modern attacks exploit a long **chain of dormant abstractions** inside deployed functional systems.
- The **composition** of these primitives can give attackers powerful programming models to program **weird machines**.

Example: DOJITA Browser Attack

$$
\begin{array}{l}
\text{while (true) } \{ \\
\quad x = input(); \\
\quad \text{if } (x == 1) \\
\quad\quad ... \\
\quad\quad *ptr(x) = f(x); \\
\quad \text{else if } (x == 2) \\
\quad\quad ... \\
\quad\quad out = IR_{ptr}; \\
\quad \text{else if } (x == 3) \\
\quad\quad ... \\
\quad\quad p = \text{jit}(IR_{ptr}) \\
\}
\end{array}
$$

Concrete Program

DOJITA Browser Attack

while (true) $\{$	$\longrightarrow$	$($
$\quad x = input();$		
$\quad$ if $(x == 1)$		
$\qquad \ldots$		$\quad \ldots$
$\qquad *ptr(x) = f(x);$	$\longrightarrow$	$\quad$ Injection
$\quad$ else if $(x == 2)$	$\longrightarrow$	$\mid$
$\qquad \ldots$		$\quad \ldots$
$\qquad out = IR_{ptr};$	$\longrightarrow$	$\quad$ Leak
$\quad$ else if $(x == 3)$	$\longrightarrow$	$\mid$
$\qquad \ldots$		$\quad \ldots$
$\qquad p =$ jit(IR_{ptr})	$\longrightarrow$	$\quad$ Jit
$\}$		$)^*$
Concrete Program	Refinement	Abstract Weird Machine

Vulnerability Flow Type System

- Existing information flow type systems enforce the correct flow of information for confidentiality and integrity but do not track vulnerabilities and their composition for higher-level malicious behavior.

- This project puts forward a new venue of investigation for novel type theories that **track unintended in addition to intended computation and flow**.

- This project will define and implement a type theory to **derive the abstract weird machines** that programs expose.

Attacks Composed of Vulnerabilities

- We will develop type systems that **derive abstract weird machines of programs as abstract control flow patterns over vulnerability types**.
- The type system tracks information flow to derive whether vulnerabilities are present, and further tracks the control flow between vulnerabilities.
- The derived weird machines can be used **to detect and disrupt attacks**.
- **Composition is the key to both a successful attack and a successful mitigation**: if the abstract program of an attack is exploiting a given sequence of vulnerabilities, sandboxing one vulnerability or reordering their flow can disrupt the attack.

DOJITA Browser Attack

Concrete Program	Refinement	Abstract Weird Machine
while (true) $\{$	$\longrightarrow$	(
$x = input();$		
if $(x == 1)$		
...		...
$*ptr(x) = f(x);$	$\longrightarrow$	Injection
else if $(x == 2)$	$\longrightarrow$	$\mid$
...		...
$out = IR_{ptr};$	$\longrightarrow$	Leak
else if $(x == 3)$	$\longrightarrow$	$\mid$
...		...
$p =$ jit(IR_{ptr})	$\longrightarrow$	Jit
$\}$		$)^*$

Abstract weird machine: (Injection | Leak | Jit)*.
It allows the DOJITA attack Leak · Injection · Jit.
Both captured as the regular expression
Any emergent behavior from the captured vulnerabilities of the concrete program is a behavior of the abstract program.

Attacks are Regular Expressions

- It further tracks the abstract flow between vulnerability types such as Leak, Injection, and Jit.
- It derives **weird machines as regular expression terms** on vulnerability types.
- Regular expressions as a **uniform description language** for exploitable weird machines.
- Regular expressions can capture **attack patterns** that are often **simple, compositional and platform independent**.
- The language **inclusion decision** for regular expressions that checks the possibility of an attack is remarkably **efficient**.

Syntax

$$
\begin{array}{rcll}
e & ::= & n \;\mid\; x \;\mid\; e_1 \oplus e_2 \;\mid\; e_1;\, e_2 & \text{Program} \\
 & \mid & \textsf{if}\ e\ e_1\ \textsf{else}\ e_2 \;\mid\; \textsf{while}\ e\ e' & \\
 & \mid & x := e \;\mid\; \textsf{jit}\ e & \\
t & ::= & w, f & \text{Types} \\
w & ::= & w \cdot w \;\mid\; w \,|\, w \;\mid\; w^* \;\mid\; \epsilon & \text{Weird Machine} \\
 & & \textsf{Leak} \;\mid\; \textsf{Injection} \;\mid\; \textsf{Jit} & \text{Vulnerability Types} \\
f & ::= & \langle c, i \rangle & \text{Information Flow Type} \\
c & ::= & \textsf{L} \;\mid\; \textsf{H} & \text{Confidentiality Type} \\
i & ::= & \textsf{L} \;\mid\; \textsf{H} & \text{Integrity Type}
\end{array}
$$

Typing Judgments

$$\Gamma, f_{\mathsf{x}} \vdash e : w, f$$

Type System

$$\textsc{Val-Type}$$
$$\Gamma, f_{\mathsf{x}} \vdash n\colon \epsilon, \langle \mathsf{L}, \mathsf{H} \rangle$$

$$\textsc{Var-Type}$$
$$\frac{\Gamma(x) = f}{\Gamma, f_{\mathsf{x}} \vdash x\colon \epsilon, f}$$

Type System

$$\textsc{Op-Type}\quad \frac{\Gamma, f_{\mathsf{x}} \vdash e : w, f \qquad \Gamma, f_{\mathsf{x}} \vdash e' : w', f'}{\Gamma, f_{\mathsf{x}} \vdash e \oplus e' : w \cdot w', f \sqcup f'}$$

Type System

$$\textsc{Seq-Type}\quad \frac{\Gamma, f_{\mathsf{x}} \vdash e : w, f \qquad \Gamma, f_{\mathsf{x}} \vdash e' : w', f'}{\Gamma, f_{\mathsf{x}} \vdash e; e' : w \cdot w', f'}$$

$$\textsc{If-Type}\quad \frac{\Gamma, f_{\mathsf{x}} \vdash e : w, f \qquad \Gamma, f_{\mathsf{x}} \sqcup f \vdash e' : w', f' \qquad \Gamma, f_{\mathsf{x}} \sqcup f \vdash e'' : w'', f''}{\Gamma, f_{\mathsf{x}} \vdash \mathsf{if}\ e\ e'\ \mathsf{else}\ e'' : w \cdot (w' \mid w''), f' \sqcup f''}$$

$$\frac{\textsc{While-Type}\qquad \Gamma, f_{\mathsf{x}} \vdash e : w, f \qquad \Gamma, f_{\mathsf{x}} \sqcup f \vdash e' : w', f'}{\Gamma, f_{\mathsf{x}} \vdash \mathsf{while}\; e\; e' : w \cdot (w' \cdot w)^{*}, f'}$$

$$\dfrac{\begin{array}{l}\textsc{Assn-Type} \\ \Gamma(x) = \langle c, i \rangle \qquad \Gamma, \langle c_{\mathsf{x}}, i_{\mathsf{x}} \rangle \vdash e \colon w, \langle c', i' \rangle \\ \qquad w' = \begin{cases} \epsilon & \text{if } c' \sqcup c_{\mathsf{x}} \sqsubseteq c \\ \mathsf{Leak} & \text{else} \end{cases} \\ \qquad w'' = \begin{cases} \epsilon & \text{if } i' \sqcup i_{\mathsf{x}} \sqsubseteq i \\ \mathsf{Injection} & \text{else} \end{cases} \end{array}}{\Gamma, \langle c_{\mathsf{x}}, i_{\mathsf{x}} \rangle \vdash x \coloneqq e \colon w \cdot w' \cdot w'', \langle c, i \rangle}$$

$$\textsc{Jit-Type}\quad \frac{\Gamma, \langle c_{\mathsf{x}}, i_{\mathsf{x}}\rangle \vdash e : w, \langle c, i\rangle \qquad w' = \begin{cases} \epsilon & \text{if } i \sqcup i_{\mathsf{x}} \sqsubseteq \mathsf{H} \\ \mathsf{Jit} & \text{else} \end{cases}}{\Gamma, \langle c_{\mathsf{x}}, i_{\mathsf{x}}\rangle \vdash \mathsf{jit}\, e : w \cdot w', \langle c, i\rangle}$$

ASSN-SEM

$$\langle \sigma, \mathcal{R}[x := n] \rangle \to$$
$$\langle \sigma[x \mapsto n], \mathcal{R}[n] \rangle$$

Instrumented Operational Semantics

Assn-ISem

$$\frac{\Gamma(x) = \langle c, i\rangle \qquad f_{\mathsf{x}} = \langle c_{\mathsf{x}}, i_{\mathsf{x}}\rangle \qquad v = \langle n, \langle c', i'\rangle\rangle \\ w_1 = \begin{cases}\epsilon & \text{if } c' \sqcup c_{\mathsf{x}} \sqsubseteq c \\ \mathsf{Leak} & \text{else}\end{cases} \qquad w_2 = \begin{cases}\epsilon & \text{if } i' \sqcup i_{\mathsf{x}} \sqsubseteq i \\ \mathsf{Injection} & \text{else}\end{cases} \qquad w = w_1 \cdot w_2}{\langle \Gamma, \gamma, f_{\mathsf{x}}, \mathcal{R}[x := v]\rangle \xrightarrow{w} \langle \Gamma, \gamma[x \mapsto v], f_{\mathsf{x}}, \mathcal{R}[v]\rangle}$$

For every execution with the operational semantics, there is a corresponding execution with the instrumented operational semantics, and vice versa.

Type-safety Guarantees

- If the type system associates a weird machine to a program, that weird machine covers the weird behavior that the executions of the program can exhibit.

Theorem 2 (Type-safety)**.** *For all* Γ*,* f_{x}*,* e w*,* f*,* γ*,* w'*,* γ'*,* f'_{x}*, and* e'*, if*

$$\Gamma, f_{\mathsf{x}} \vdash e : w, f,$$

$$\Gamma \models \gamma, \text{ and}$$

$$\langle \Gamma, \gamma, f_{\mathsf{x}}, e \rangle \xrightarrow{w'}{}^{*} \langle \Gamma, \gamma', f'_{\mathsf{x}}, e' \rangle,$$

then

$$w' \Subset w.$$

Type-safety Guarantees

- If the type system type-checks a program e as the weird machine w, and w does not intersect with an attack pattern w′, then no execution of the program can produce an instance of that attack.

Corollary 2.1. *For all* Γ, f_{x}, e w, f, γ, w', γ', f'_{x}, *and* e',
if

$$\Gamma, f_{\mathsf{x}} \vdash e : w, f,$$
$$\Gamma \models \gamma,$$
$$w \cap w' = \emptyset, \textit{ and}$$
$$w'' \subseteq w',$$

then

$$\langle \Gamma, \gamma, f_{\mathsf{x}}, e \rangle \overset{w''}{\not\to}{}^{*} \langle \Gamma, \gamma', f'_{\mathsf{x}}, e' \rangle.$$

Conclusion

- The type system can detect attacks that compose multiple vulnerabilities such as the DOJITA browser attack.
- The checker that can **detect the presence and absence of composed attacks** has the promise to be adopted by many who strive for more secure software.

A few composed attacks

KOOBE: Towards Facilitating Exploit Generation of Kernel Out-Of-Bounds Write Vulnerabilities. USENIX Security 20

JITGuard. Hardening Just-in-time Compilers with SGX. CCS 2017. T. Frassetto, D. Gens, C. Liebchen, A.-R. Sadeghi

Towards Automated Generation of Exploitation Primitives for Web Browsers. ACSAC ’18

Speculative Probing: Hacking Blind in the Spectre Era. CCS 2020.