Compiler-instrumented, Dynamic Secret-Redaction of Legacy Processes for Attacker Deception

Frederico Araujo and Kevin Hamlen

*The University of Texas at Dallas*
Problem: Live Process Image Secret-Redaction

• Goal:
  – Remove or replace secrets in address spaces of *running programs*, yielding processes that **CONTINUE RUNNING** (but with no secrets)

• Potential Applications:
  – Debugging: Safely disclose redacted crash dumps to developers
  – Intrusion Response: Dynamic secret redaction without loss of service
  – Cyber Deception: Runtime replacement of secrets with honey-data
    • *Honey-patching [CCS’14]*
Problem: Live Process Image Secret-Redaction

- **Goal:**
  - Remove or replace secrets in address spaces of *running programs*, yielding processes that **CONTINUE RUNNING** (but with no secrets)

- **Potential Applications:**
  - **Debugging:** Safely disclose redacted crash dumps to developers
  - **Intrusion Response:** Dynamic secret redaction without loss of service
  - **Cyber Deception:** Runtime replacement of secrets with honey-data
    - *Honey-patching [CCS’14]*
Runtime Secret Redaction

```
GET / HTTP/1.1 /browse/doc1.html
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US, en; q=0.8
Cookie: app.token=BACC-76GF-ABS3-ZOV274f89abc43de7
SESSIONID=2321CFA5DA771A284D13DD67798A7ED1D554E
Accept-Language: en
Accept:text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:32.0) Gecko/20100101 Firefox/32.0
GET / HTTP/1.1 /browse/doc1.html
```

Diagram:
- Web server (honey-patched)
- Web server (unpatched clone)
- Controller
- Reverse proxy
- Container pool
- 1. Request
- 2. Request
- 3. Fork
- 4. Clone
- 5. Response
- 6. Response

Attacker deception process:
1. Request
2. Request
3. Fork
4. Clone
5. Response
6. Response

User 1
User 2
Attacker
Main idea: *Instrument programs with operations that track (explicit) dataflows of secrets.*

- Program vulnerability detection
  - TaintCheck, LIFT, Mimemu, Argos, ...
- Information leak detection
  - TaintDroid, TaintEraser, AndroidLeaks, Spandex, D2Taint, ...
- Study of sensitive data lifetime
  - TaintBochs
- Analysis of spyware behavior
  - Panorama, Hookfinder, PHP Aspis, ...
- Test set generation
  - Memsherlock, ConfAid, ...
Dynamic Taint Tracking

```c
/* first colon delimits username:password */
s1 = memchr(hostinfo, ':', s - hostinfo);
if (s1) {
    uptr->user = memdup(hostinfo, s1 - hostinfo);
    ++s1;
    uptr->password = memdup(s1, s - s1);
}

void safe_free(char *s) {
    if (s is a secret) // how to test whether s is secret?
        slow_secure_free(s);
    else
        free(s);
}
```
Taint Introduction:

```c
/* first colon delimits username:password */
s1 = memchr(hostinfo, ':', s - hostinfo);
if (s1) {
    uptr->user = memdup(hostinfo, s1 - hostinfo);
    ++s1;
    uptr->password = memdup(s1, s - s1);
    dfsan_set_label(SECRET, uptr->password, sizeof(s - s1))
}
```

```c
void safe_free(char *s) {
    if (s is a secret) // how to test whether s is secret?
        slow_secure_free(s);
    else
        free(s);
}
```
Dynamic Taint Tracking

Taint Introduction:

```c
/* first colon delimits username:password */
s1 = memchr(hostinfo, ':', s - hostinfo);
if (s1) {
    uptr->user = memdup(hostinfo, s1 - hostinfo);
    ++s1;
    uptr->password = memdup(s1, s - s1);
    dfsan_set_label(SECRET, uptr->password, sizeof(s - s1))
}
```

Taint Check:

```c
void safe_free(char *s) {
    if (dfsan_get_label(s) == SECRET)
        slow_secure_free(s);
    else
        free(s);
}
```
Retrofitting Headaches

Taint Introduction:

```c
/* first colon delimits username:password */
s1 = memchr(hostinfo, ':', s - hostinfo);
if (s1) {
    uptr->user = memdup(hostinfo, s1 - hostinfo);
    ++s1;
    uptr->password = memdup(s1, s - s1);
    dfsan_set_label(SECRET, uptr->password, sizeof(s - s1))
}
```

...
Retrofitting Headaches

Taint Introduction:

```c
p = &(uptr->password);
*p = malloc(...);
...
/* first colon delimits username:password */
s1 = memchr(hostinfo, ':', s - hostinfo);
if (s1) {
    uptr->user = memdup(hostinfo, s1 - hostinfo);
    ++s1;
    *p = memdup(s1, s - s1);
    dfsan_set_label(SECRET, *p, sizeof(s - s1))
}
...
```

Apache is 2.2M SLOC!
Our Solution: Declarative Secret Annotations for C

- **Declarative vs. Operational Secret Annotations**
  - Fewer declarations than operations for user to annotate
  - Compiler infers and implements operations from declarations
  - Compiler optimizes operational implementation

- **SECRET** = struct contains secrets
- **SECRET_STR** = field is a pointer to a null-terminated sequence of secret chars
- **NONSECRET** = field is a non-secret within a SECRET struct

```
struct apr_uri_t {
    NONSECRET char *user;
    SECRET_STR char *password;
    ...
} SECRET;
```
SignaC: Secret Information Graph iNstrumentation for Annotated C

Compiler-instrumented, Dynamic Secret-Redaction of Legacy Processes for Attacker Deception

- **Clang/LLVM**: Clang preprocessor rewriting, LLVM pass taint-tracking.
- **Source Code**: Clang/LLVM source code instrumentation.
- **Type Annotations**: SECRET, ...
- **PC2S**: taint-tracking binary.
Outline

1. Overview
2. Challenges & Approach
3. Implementation
4. Evaluation
5. Application Study
6. Conclusions
Taint Propagation

\[
p = \&(uptr->password); \\
*p = malloc(...);
\]

...  

/* first colon delimits username:password */  
s1 = memchr(hostinfo, ':', s – hostinfo);  
if (s1) {  
    uptr->user = memdup(hostinfo, s1 – hostinfo);  
    ++s1;  
    memcpy(*p, s1, s – s1);  
}  

...  

Need taint propagation semantics for...  
• field access operator (->)  
• address-of (&) operator  
• assignments (=)  
• dereferencing assignments (*p = ...)  
• dynamic memory allocations (malloc)
Pointer Combine Semantics (PCS)

mytype *p = ...;
mytype v = ...;
dfsan_set_label(USER1, p, sizeof(p));
dfsan_set_label(USER2, v, sizeof(v));
*p = v;

What should be the resulting label of *p?

- **Two standard answers:**
  - **No-Combine Semantics:** label of *p is just USER2
    - Rationale: v was copied; its ownership didn’t change.
  - **Combine Semantics:** label of *p is USER1 \(\sqcup\) USER2 (joint ownership)
    - Rationale: Failing to redact value at *p now possibly divulges value of pointer p.
    - Conclusion: Value *p is now one of USER1’s secrets (as well as continuing to be one of USER2’s secrets).
**Pointer Combine Semantics (PCS)**

**propagation semantics:**

\[
\begin{align*}
\gamma_p &\rightarrow \gamma_p \uplus \gamma_v \\
\gamma_p &\rightarrow \gamma_v \\
\gamma_v &\rightarrow \gamma_v \\
\end{align*}
\]

- \( p = \& (\text{uptr} \rightarrow \text{password}); \)
- \( *p = v \)
- \( *p = \text{malloc}(...) \)
- \( *p[i] = v \) (for all \( i \))
- \( x = *p[i] \) (for all \( i \))

**revisiting our initial example...**

```c
struct apr_uri_t {
    NONSECRET char *user;
    SECRET_STR char *password;
    ...
} SECRET;
```

- \( \text{uptr} \rightarrow \text{user} = v1; \)
- \( \text{uptr} \rightarrow \text{password} = v2; \)
PCS Problems: Overtainting & Label Creep

but...

very common...

```
while (freelist != NULL) {
    node = freelist;
    freelist = node->next;
    free(node);
}
```

```
while (prev) {
    prev->eos_sent = 1;
    prev = prev->prev;
}
```
Our Solution: Pointer Conditional-Combine Semantics ($PC^2S$)

```c
mytype *p = ...;
mytype v = ...;
dfSAN_set_label(USER1, p, sizeof(p));
dfSAN_set_label(USER2, v, sizeof(v));
*p = v;
```

What should be the resulting label of $*p$?

A New Third Answer:

- **Conditional-Combine Semantics**: label of $*p$ depends upon the *static type* of $v$!
  - if $v$ has *pointer* type, then use *No-Combine Semantics* (USER2).
  - if $v$ has *non-pointer* type, then use *Combine Semantics* (USER1 $\sqcup$ USER2).
Our Solution: Pointer Conditional-Combine Semantics (PC²S)

Propagation semantics:

value-to-pointer store

pointer-to-pointer store

taint policy: “do not combine when pointee has pointer type”

let’s try again…

Compiler-instrumented, Dynamic Secret-Redaction of Legacy Processes for Attacker Deception
Outline

1. Overview
2. Challenges & Approach
3. Implementation
4. Evaluation
5. Application Study
6. Conclusions
SignaC: Type Attributes

Annotated Types

```c
struct request_rec {
    NONSECRET ... *pool;
    apr_uri_t parsed_uri;
    ...
} SECRET;
```

Rewriting

```c
clang transformation

new = (request_rec *) apr_pcallo(r->pool, ...);

new = (request_rec *) signac_alloc(apr_pcallo, r->pool, ...);
```

Instrumentation

```c
clang/LLVM -dfsan -pc2s
```

# define SECRET __attribute__((annotate("secret")))

### clang -Xclang -ast-dump

```
| -RecordDecl 0x8943a40 <line:15:9, line:19:1> line:15:16 struct request_rec definition |
| -AnnotateAttr 0x8943d60 <line:3:31, col:48> "secret"
| -FieldDecl 0x8943c20 <line:16:5, col:29> col:29 referenced pool 'apr_pool_t * ...
| ` -FieldDecl 0x8943ca0 <line:17:5, col:15> col:15 parsed_uri 'apr_uri_t': 'struct apr_uri_t`
```
### SignaC: Type Attributes

#### Annotated Types

```c
struct request_rec {
   NONSECRET ... *pool;
   apr_uri_t parsed_uri;
   ...
} SECRET;
```

#### Rewriting

```c
clang transformation

new = (request_rec *) apr_pcallo(r->pool, ...);
new = (request_rec *) signac_alloc(apr_pcallo, r->pool, ...);
```

#### Quala Type Qualifiers

```c
#define NONSECRET __attribute__((type_annotate("nonsecret")))
#define SECRET_STR __attribute__((type_annotate("secret_str")))
```

#### Instrumentation

```c
clang/LLVM -dfsan -pc2's
instrumented binary
libsinaC
```

#### Code Snippet

```c
rec->pool = pool;
```

```c
%3 = load %struct.apr_pool_t** %pool, align 8
%4 = load %struct.request_rec** %rec, align 8
%pool2 = getelementptr inbounds %struct.request_rec* %4, i32 0, i32 0
store %struct.apr_pool_t* %3, %struct.apr_pool_t** %pool2, align 8, !tyann !1
...!

!1 = !{"nonsecret", i8 0}
```
SignaC: Type Attribute Rewriting

→ Clang tooling API: AST Matchers + Rewriting API
→ Allocators list: {malloc, calloc, apr_palloc, apr_palloc, ...}

1. Compute the list of all **security-relevant datatypes**

```c
# define signac_alloc(alloct, args...) {{
    void * __p = alloc( args);
    signac_taint(__p, sizeof(void*));
    __p; }}
```

2. Wrap all security-relevant datatypes instantiations with `signac_alloc`
Annotated Types

```c
struct request_rec {
   NONSECRET ... *pool;
apr_uri_t parsed_uri;
   ...
} SECRET;
```

Rewriting

```c
c clang transformation
   new = (request_rec *) apr_palloc(r->pool, ...);

new = (request_rec *) signac_alloc(apr_palloc, r->pool, ...);
```

Instrumentation

```c
clang/LLVM -dfsan -pc2s
```

→ implemented as an extension to DFSan
→ low-overhead representation of labels: 16-bit integers allocated sequentially
→ maps without reserving the lower 32TB of the process address space for shadow memory
→ union labels organized as a dynamically growing binary (DAG) – the union table

<table>
<thead>
<tr>
<th>Start</th>
<th>End</th>
<th>Memory Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x700000008000</td>
<td>0x8000000000000000</td>
<td>application memory</td>
</tr>
<tr>
<td>0x200000000000</td>
<td>0x700000008000</td>
<td>union table</td>
</tr>
<tr>
<td>0x000000100000</td>
<td>0x2000000000000000</td>
<td>shadow memory</td>
</tr>
<tr>
<td>0x000000000000</td>
<td>0x000000100000</td>
<td>reserved by kernel</td>
</tr>
</tbody>
</table>

example: $Z = X + Y$

Grammar example:

1. $X = 1$
2. $Y = 2$
3. $Z = X + Y$
SignaC: PC²S Instrumentation

→ label propagation across external library interfaces expressed as an **ABI list**
→ DFSan predefines an ABI list that covers glibc

fun:malloc=custom
fun:realloc=discard
fun:free=discard
... fun:isalpha=functional
fun:isdigit=functional
... fun:memcpy=custom
fun:memset=custom
fun:strcpy=custom

**discard** $\rho_{\text{dis}}(\overline{V}) := \bot$

**functional** $\rho_{\text{fun}}(\overline{V}) := \bigcup \overline{V}$

**custom** custom-defined label propagation wrapper

→ other libraries mapped to the ABI: OpenSSL, PCRE, APR, ...
→ **memory transfer functions** (e.g., strcpy, strdup) and **input functions** (e.g., read, pread) ABI extensions for PC²S
SignaC: PC²S Instrumentation

→ instrumentation operates on LLVM IR, inserting label propagation code
→ propagation policy parametrized at the compiler’s front-end: \textit{pc2s-on-store}, \textit{pc2s-on-load}

\textbf{Example:}

\textbf{Store instruction}

```c
struct request_rec {
    NONSECRET ... *pool;
    apr_uri_t parsed_uri;
    ...
} SECRET;
```

<table>
<thead>
<tr>
<th>Annotated Types</th>
<th>Rewriting</th>
<th>Instrumentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbf{IR}</td>
<td>clang transformation</td>
<td>\textbf{IR}</td>
</tr>
<tr>
<td>ptr ← SI.getPointerOperand()</td>
<td>value ← SI.getValueOperand()</td>
<td>shadow ← getShadow(value)</td>
</tr>
<tr>
<td>storeShadow(ptr, ..., shadow, &amp;SI)</td>
<td>isSecretStr(&amp;SI)</td>
<td>str ← createBitCast(value)</td>
</tr>
</tbody>
</table>

check for non-SECRET annotated pointers
special handling of \textbf{SECRET STR} annotated types
SignaC: Runtime Library

Annotated Types

```
struct request_rec {
   #ifndef NONSECRET
   ... *
   
   #endif
   pool;
   apr_uri_t parsed_uri;
   ...
} SECRET;
```

Rewriting

```
clang transformation

new = (request_rec *) apr_pcalloc(r->pool, ...);

new = (request_rec *) signac_alloc(apr_pcalloc, r->pool, ...);
```

Instrumentation

```
clang/LLVM -dfsan -pc2s
instrumented
binary
libsignaC
```

→ libsignaC: tiny C library that encapsulates runtime support for the type annotation mechanism

```
signac_init(pl)
```
initialize a tainting context with a fresh label instantiation `pl` for the current principal.

```
signac_taint(addr, size)
taint each address in interval `[addr; addr+size)` with `pl`.
```

```
signac_alloc(alloc, ...)
wrap allocator alloc and taint the address of its returned pointer with `pl`.
```
Label Creep

![Graph showing the number of labels over requests]

- PC$^2$S
- PCS

requests: 68
Taint Spread

![Taint Spread Graph]

- Compiler-instrumented, Dynamic Secret-Redaction of Legacy Processes for Attacker Deception

- PC$^2$S (solid line)
- PCS (dashed line)

- Tainted bytes (kB) vs. requests

- Key observation: Taint spread is significantly reduced compared to the baseline (redacted to prevent disclosure).

- Request threshold for significant taint spread: < 3 (redaction)
Table 2: Average overhead of instrumentation

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>$C=1$</th>
<th>$C=10$</th>
<th>$C=50$</th>
<th>$C=100$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>2.50</td>
<td>2.34</td>
<td>2.56</td>
<td>2.32</td>
</tr>
<tr>
<td>CGI Bash</td>
<td>1.29</td>
<td>0.98</td>
<td>1.00</td>
<td>0.97</td>
</tr>
<tr>
<td>PHP</td>
<td>0.41</td>
<td>0.37</td>
<td>0.30</td>
<td>0.31</td>
</tr>
</tbody>
</table>
Compiler-instrumented, Dynamic Secret-Redaction of Legacy Processes for Attacker Deception

Outline

1. Overview
2. Challenges & Approach
3. Implementation
4. Evaluation
5. Application Study
6. Conclusions
Compiler-instrumented, Dynamic Secret-Redaction of Legacy Processes for Attacker Deception
Secret Redaction Performance

Compiler-instrumented, Dynamic Secret-Redaction of Legacy Processes for Attacker Deception
Deception Strategy: Artificially delay non-forking responses to match the forking delay.
Validation Tests

Table 1: Honey-patched security vulnerabilities

<table>
<thead>
<tr>
<th>Software</th>
<th>Version</th>
<th>CVE-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>2.2.21</td>
<td>CVE-2011-3368</td>
<td>Improper URL Validation</td>
</tr>
<tr>
<td></td>
<td>2.2.9</td>
<td>CVE-2010-2791</td>
<td>Improper timeouts of keep-alive connections</td>
</tr>
<tr>
<td></td>
<td>2.2.15</td>
<td>CVE-2010-1452</td>
<td>Bad request handling</td>
</tr>
<tr>
<td></td>
<td>2.2.11</td>
<td>CVE-2009-1890</td>
<td>Request content length out of bounds</td>
</tr>
<tr>
<td></td>
<td>2.0.55</td>
<td>CVE-2005-3357</td>
<td>Bad SSL protocol check</td>
</tr>
<tr>
<td>OpenSSL</td>
<td>1.0.1f</td>
<td>CVE-2014-0160</td>
<td>Buffer over-read in heartbeat extension</td>
</tr>
<tr>
<td>Bash</td>
<td>4.3</td>
<td>CVE-2014-6271</td>
<td>Improper parsing of environment variables</td>
</tr>
</tbody>
</table>
Outline

1. Overview
2. Challenges & Approach
3. Implementation
4. Evaluation
5. Application Study
6. Conclusions
Conclusions

• Declarative annotation of secrets
  – New pointer tainting methodology
  – Reduced secret annotation burden

• New taint propagation semantics
  – Accurate while containing taint spread and label creep
  – Implemented in LLVM

• Implemented a memory redactor for secure honey-patching

• Tested on three production web servers
Thank you!
Questions?

Frederico Araujo
(frederico.araujo@utdallas.edu)
Operational Semantics

\[
\begin{align*}
\text{programs} & \quad \mathcal{P} ::= \varepsilon \\
\text{commands} & \quad e ::= \text{v} := e \mid \text{store}(\tau, e_1, e_2) \mid \text{ret}(\tau, e) \\
& \quad \quad \mid \text{call}(\tau, e, \text{args}) \mid \text{br}(e, e_1, e_0) \\
\text{expressions} & \quad e ::= \text{v} \mid \langle u, \gamma \rangle \mid \Diamond_b(\tau, e_1, e_2) \mid \text{load}(\tau, e) \\
\text{binary ops} & \quad \Diamond_b ::= \text{typical binary operators} \\
\text{variables} & \quad \nu \\
\text{values} & \quad u ::= \text{values of underlying IR language} \\
\text{types} & \quad \tau ::= \text{ptr}\,\tau \mid \tau \mid \tau\,\tau \mid \text{primitive types} \\
\text{label types} & \quad \gamma \in (\Gamma, \square) \quad \text{(label lattice)} \\
\text{locations} & \quad \ell ::= \text{memory addresses} \\
\text{environment} & \quad \Delta : \nu \rightarrow u \\
\text{prog counter} & \quad \text{pc} \\
\text{stores} & \quad \sigma : (\ell \rightarrow u) \cup (\nu \rightarrow \ell) \\
\text{functions} & \quad f \\
\text{function table} & \quad \phi : f \rightarrow \ell \\
\text{tainu context} & \quad \lambda : (\ell \cup \nu) \rightarrow \gamma \\
\text{propagation} & \quad \rho : \tau \rightarrow \gamma \\
\text{prop context} & \quad \Lambda : f \rightarrow \rho \\
\text{call stack} & \quad \Xi ::= \text{nil} \mid (f, \text{pc}, \Delta, \tau) :: \Xi
\end{align*}
\]

Figure 2: Intermediate representation syntax.

NCS \[\rho_{\text{load, store}}(\tau, \gamma_1, \gamma_2) := \gamma_2\]
PCS \[\rho_{\text{load, store}}(\tau, \gamma_1, \gamma_2) := \gamma_1 \sqcup \gamma_2\]
PC\textsuperscript{2}S \[\rho_{\text{load, store}}(\tau, \gamma_1, \gamma_2) := (\text{if } f \text{ is ptr} ? \gamma_2 : (\gamma_1 \sqcup \gamma_2))\]

Figure 4: Polymorphic functions modeling no-combine, pointer-combine, and PC\textsuperscript{2}S label propagation policies.
Compiler-instrumented, Dynamic Secret-Redaction of Legacy Processes for Attacker Deception