Wyvern: Impacting Software Security via Programming Language Design

Darya Kurilova*   Alex Potanin^   Jonathan Aldrich*

* Carnegie Mellon University
^ Victoria University of Wellington
Data Breaches Compromised Millions of User Records Since 2009

- Hacking: 236.0M records
- Accidental: 211.2M
- Physical loss: 97.3M
- Fraud: 7.0M
- Unknown: 2.5M
- Insider: 1.2M

Data breaches caused by hacked entry and malware have exposed more user records than any other cause. The attack on Target was the second largest since 2009.

© Darya Kurilova 2014
Wyvern programming language’s design solved the problem of command injection vulnerabilities.
Command Injection Attacks

- Occurs when maliciously crafted user input is executed as a command by the software system.

  E.g.:

  ```java
  String keyword = System.in.read();
  String webpage =
  "<html>" +
  "<body>" +
  "<h1>Results for " + keyword + "</h1>" +
  "<ul id="results">" + to_string(query_results(db, parse_sql(
  "SELECT title, snippet FROM products WHERE title = '" + keyword + "}
  ))) +
  "</ul"></body></html>"
  ```

**Cross-site scripting (XSS) attack:** Provide "<script>malicious_code</script>"

**SQL injection attack:** Provide "/; DROP TABLE products --"
Question

Why are command injection vulnerabilities introduced?
Hypothesis 1

**H1**: Command injection vulnerabilities are often introduced because it’s easier to compose queries by pasting together strings than to use secure libraries, such as prepared SQL statements.

In other words, command injection vulnerabilities have to do with psychological acceptability.
Hypothesis 1

Status quo version:

```ml
let webpage : HTML = parse_html(  
"<html>
<body>
<h1>Results for " + keyword + "</h1>
<ul id="results">
  to_string(query_results(db, parse_sql("  
    SELECT title, snippet FROM products WHERE title = '" + keyword + "'")))) +  
"</ul></body></html>"
)
```

Safe, library-based version:

```ml
let webpage : HTML = HTMLElement(Dict.empty(), [BodyElement(Dict.empty(),  
  [H1Element(Dict.empty(), [TextNode("Results for " + keyword)]),  
  ULElement(Dict.add(Dict.empty(), ("id", "results"))), query_results(db,  
    SelectStmt(["title", "snippet"], "products",  
      [WhereClause(Equals("title", StringLit(keyword))])))])])
```
Hypotheses 2 and 3

**H2:** If SQL is embedded natively in the host language, it will be easier to use SQL directly than to construct queries with strings.

**H3:** If developers use a host language’s natively embedded, safe version of SQL, they will be less likely to introduce SQL injection vulnerabilities.

In other words:
- Provide safe defaults,
- Provide positive security model, and
- Keep security simple.
Hypotheses 2 and 3

Status quo version:

```ocaml
let webpage : HTML = parse_html(
  "<html>" +
  "<body>" +
  "<h1>Results for " + keyword + "</h1>" +
  "<ul id="results">" + to_string(query_results(db, parse_sql(
    "SELECT title, snippet FROM products WHERE title = " + keyword + "'")))) +
  "</ul></body></html>"
)
```

Wyvern version*:

```ocaml
let keyword : String = user_input
let webpage : HTML = ~
  <html>
  <body>
    <h1>Results for {HTML.TextNode(keyword)}</h1>
    <ul id="results">
      {query_results(db, ~)}
      SELECT title, snippet FROM products WHERE title = {keyword}
    </ul>
  </body></html>
```


© Darya Kurilova 2014
Conclusions

• Wyvern programming language’s design addresses command injection attacks.

• More generally, programming language design can indeed impact the security of the software written in it.