



SECURE COMMUNICATIONS AT THE SPEED OF CYBER

Application Auditing

A dark green silhouette of a person's head and shoulders is positioned in the center-right of the slide. The background behind the silhouette is a light green network graph with numerous small yellow dots connected by thin lines, representing a complex system or communication network.

Suresh C. Kothari
Richardson Professor
Department of Electrical and Computer Engineering

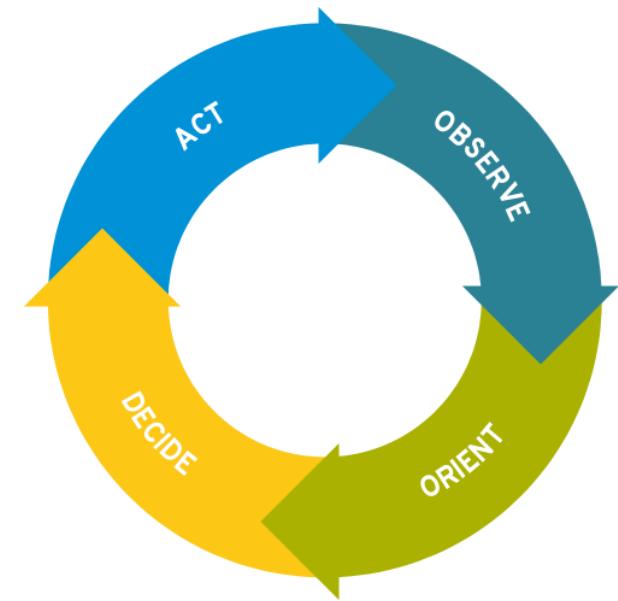
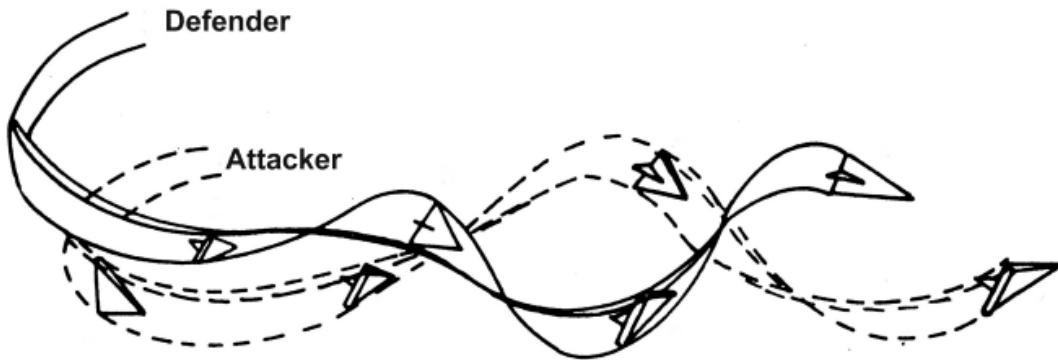
Ben Holland, Iowa State University

Acknowledgement: Team members at Iowa State University and EnSoft, DARPA contracts FA8750-12-2-0126 & FA8750-15-2-0080

BALTIMORE, MD • NOVEMBER 1–3, 2016

John Boyd's OODA Loop

“Security is a process, not a product” – Bruce Schneier





Our opponent

- Time!
- Our adversaries are looking too...

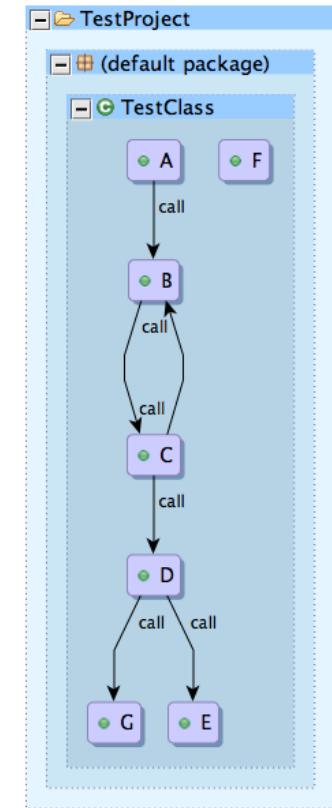
“...IA > AI, that is, that intelligence amplifying systems can, at any given level of available systems technology, beat AI systems. That is, a machine and a mind can beat a mind-imitating machine working by itself.”
– Fred Brooks

Speeding through OODA with Atlas

```
1 public class TestClass {  
2       
3     public void A() {  
4         B();  
5     }  
6       
7     public void B() {  
8         C();  
9     }  
10      
11    public void C() {  
12        B();  
13        D();  
14    }  
15      
16    public void D() {  
17        E();  
18        F();  
19    }  
20      
21    public void E() {  
22    }  
23      
24    public void F() {  
25    }  
26      
27    public void G(){  
28    }  
29      
30    }  
31      
32    }  
33 }
```

Program Declarations, Control Flow, and Data Flow

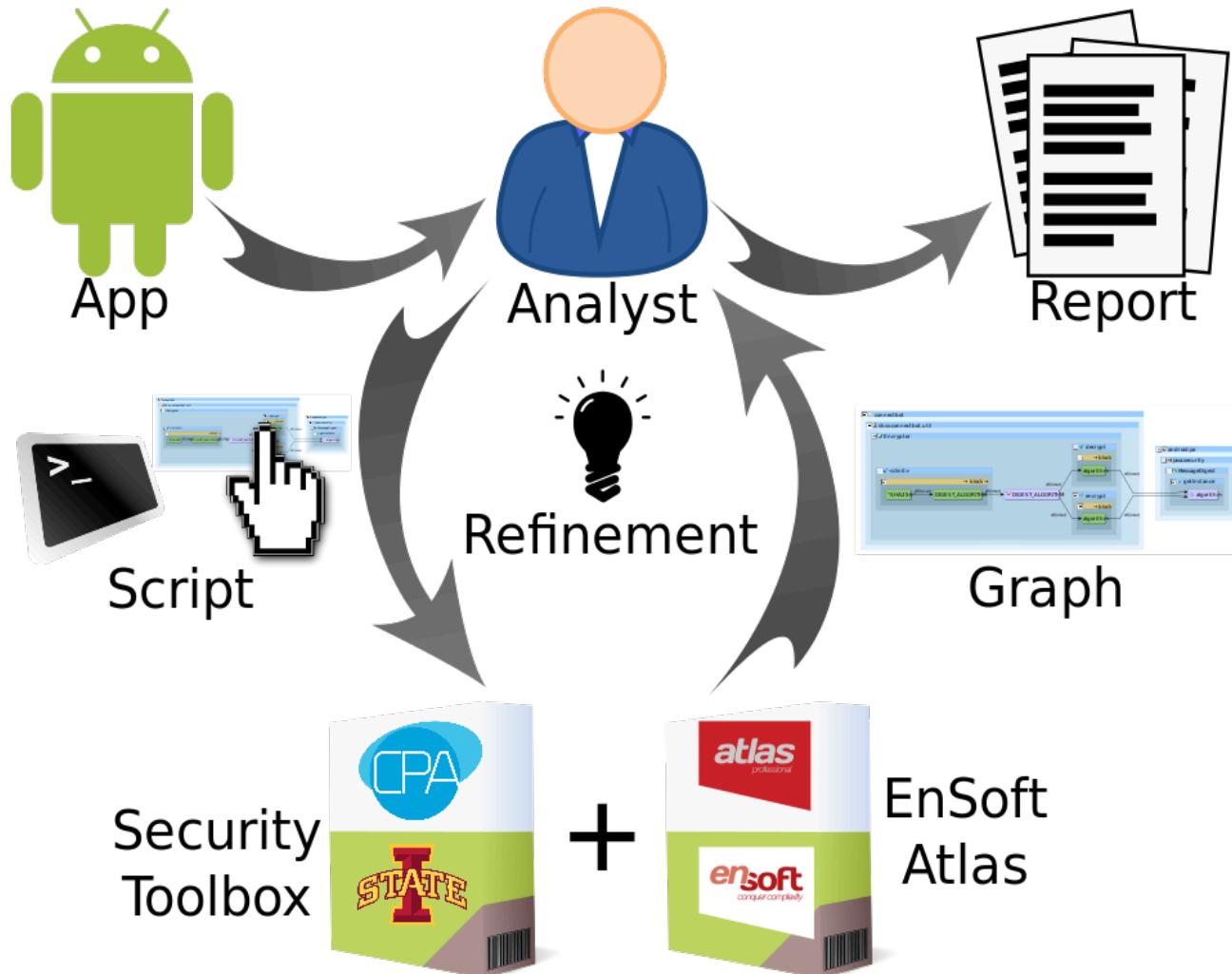
Queryable Graph Database
2-way Source Correspondence



MILCOM 2016

SECURE COMMUNICATIONS AT THE SPEED OF CYBER

BALTIMORE, MD • NOVEMBER 1–3, 2016



Lab: Auditing Web Application

- Can you find the information leakage?
- Compiled Tomcat JSP web application with SQL database
 - <https://github.com/benjholla/LoginSideChannels>
- Atlas WAR Decompiler Frontend
 - <https://ben-holland.com/AtlasWBP/>

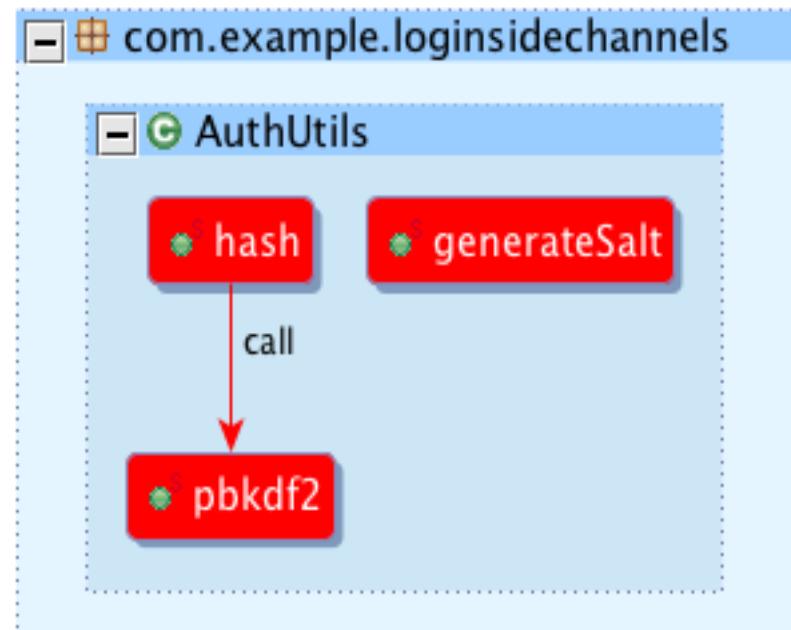
Threat Assessment

- What secrets must be protected?
- What observables does the attacker control?
- Is user enumeration a concern?
 - Consider: Ashley Madison breach

LoginSideChannels Vulnerability

- The existence of users can be inferred through timing differentials.
- More time is required to validate a password of a valid user than an invalid user.
- Attacker does not need to know any valid passwords and only has to guess at valid users.

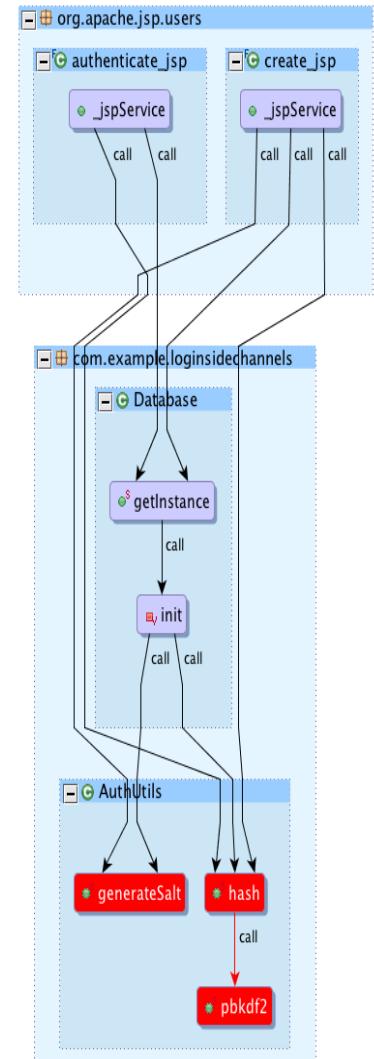
- **Approximation:** Loops are expensive and nested loops are more expensive than non-nested loops
- **Loop Call Graph:** Recovers loops, induces call edges, highlights calls of loops called within loops.
- **Note:** Hashes are computed in a feedback loop of N rounds for improved resistance to brute force attacks.



Question: Where are these loops used and why?

Analysis: Inspect call graph to get some context

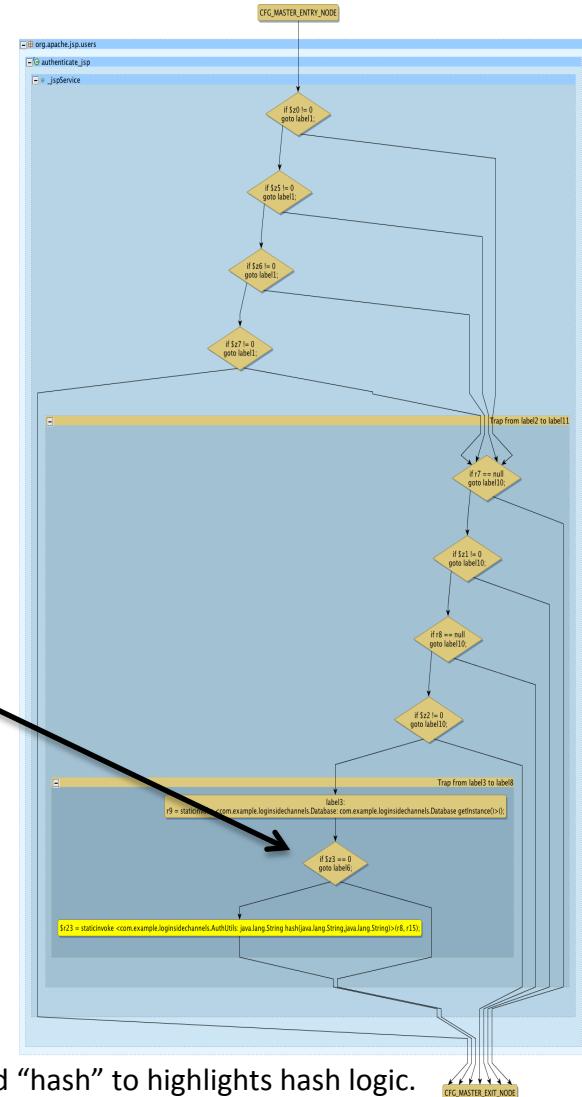
Answer: Primarily used by two services: authenticate user and create user.



Question: Is the expensive logic used conditionally?

Analysis: Compute an Event Flow Graph (EFG, a compact graph containing only relevant conditions). Inspect “authenticate_jsp” method in a EFG.

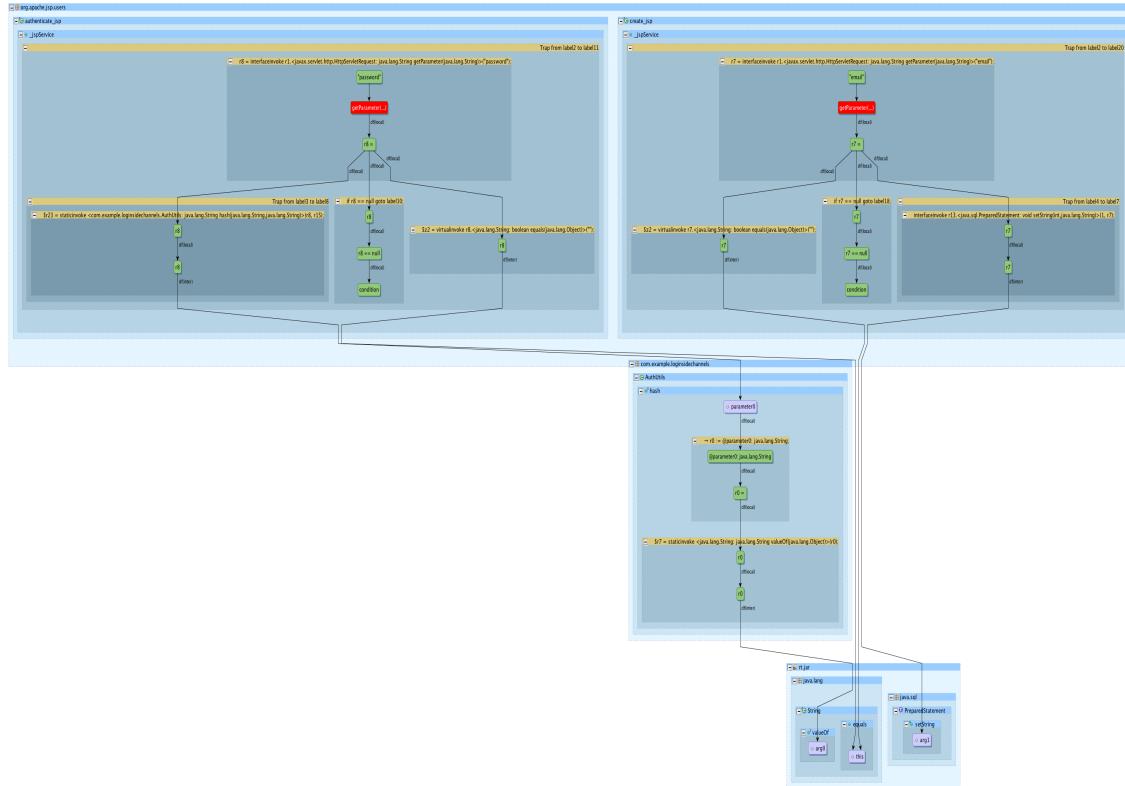
Answer: EFG reveals a conditional guard on the hash. Analyst clicks to view code. Condition depends on result of SQL query.



Question: Can a secret be deduced by this potential timing difference?

Analysis: Follow data flow forward from secrets (email, password) to conditionals.

Observation: Password flows to hash; email flows into SQL.



```
r9 = staticinvoke <Database: Database getInstance()>();  
r10 = virtualinvoke r9.<Database: Connection getConnection()>();  
r11 = interfaceinvoke r10.<Connection: PreparedStatement prepareStatement(String)>(  
    "SELECT * FROM webdb.users where Email=? LIMIT 1");  
interfaceinvoke r11.<PreparedStatement: void setString(int, String)>(1, r7);  
r12 = interfaceinvoke r11.<PreparedStatement: ResultSet executeQuery()>();  
$z3 = interfaceinvoke r12.<ResultSet: boolean next()>();  
if $z3 == 0 goto label06;  
...  
$r23 = staticinvoke <AuthUtils: String hash(String, String)>(r8, r15);
```

Observation: The SQL query controls the condition of interest.

Answer: Relatively expensive logic (hash) is invoked only if email exists in the database.

Demo: Proof of Concept Exploit

Video: user_enumeration_timing_attack.mp4

The screenshot shows the Burp Suite Free Edition v1.6 interface. The title bar reads "Burp Suite Free Edition v1.6". The menu bar includes "Burp", "Intruder", "Repeater", "Window", and "Help". The top navigation bar has tabs for "Target", "Proxy", "Spider", "Scanner", "Intruder" (which is selected), "Repeater", "Sequencer", "Decoder", "Comparer", "Extender", "Options", and "Alerts". Below the tabs are buttons for "Target", "Positions", "Payloads", and "Options".

The main window displays an "Intruder attack 1" session. The left sidebar shows "Payload Decisions" and "Attack". The right pane shows a table titled "Results" with columns: Request, Payload, Status, Response, Error, Timeout, Length, and Comment. A note "baseline request" is visible next to row 8.

The table data is as follows:

Request	Payload	Status	Response	Error	Timeout	Length	Comment
3	ben@mail.com	302	7			221	
2	jimmy@gmail.com	302	8			221	
6	linda@mail.com	302	10			221	
4	michael@mail.com	302	12			221	
5	amber@mail.com	302	13			221	
0		302	15			221	
7	obama@whitehouse.gov	302	45			221	
1	admin@example.com	302	50			221	
8	test@test.com	302	71			221	baseline request

Below the table, there are tabs for "Request" and "Response", and buttons for "Raw", "Params", "Headers", and "Hex". A search bar at the bottom says "Type a search term" with "0 matches". The status bar at the bottom right says "1 payload position" and "Length: 520".