CERIAS

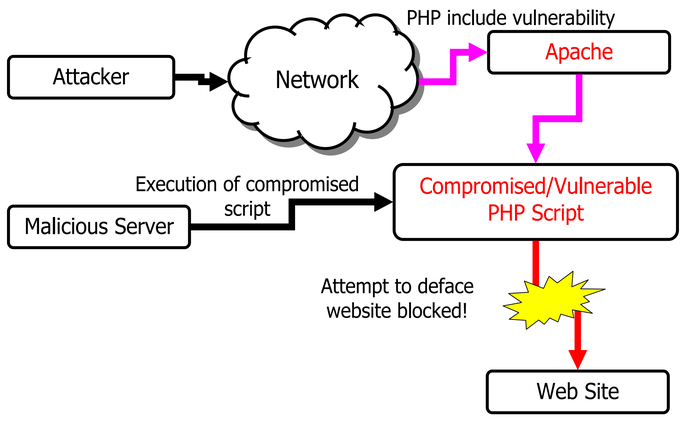
Protecting the Integrity of Networked Resources:

Taint Tracking of Files and Processes

Pascal Meunier Eric Wulf

Display Legend:

Scenario 1: PHP Remote File-Inclusion Attack



Scenario 1: Details

1> Attacker exploits remote file inclusion vulnerability to compromise an existing, legitimate PHP script. Apache becomes tainted due to receipt of network input
2> A malicious server requests execution of the compromised script, in order to use it to deface the owner's website
3> Apache executes script, and attempts to overwrite the contents of the webpage; write calls are blocked due to the webpage's untainted status!

Rules For Taint Propagation

1> Any untainted object that reads from a tainted object will become tainted itself

2> Nothing may execute a tainted object

3> A tainted object may not write to an untainted object

4> All inbound network traffic is tainted

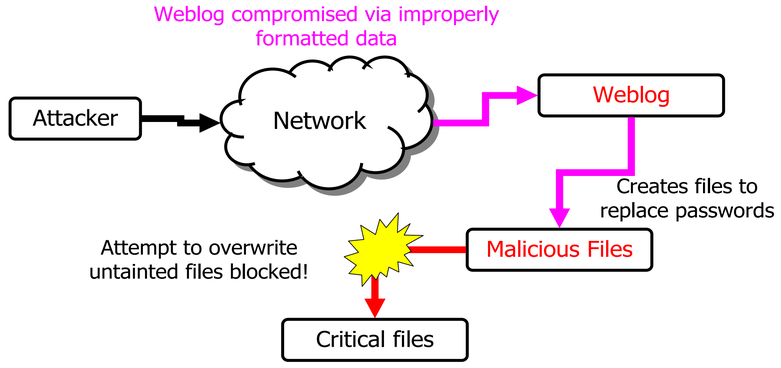
5> A remote user's login terminal cannot become tainted

6> An untainted object can change the taint status of any object owned by the same user

Taint mechanism design based on a modified Biba integrity model

The LOCUS distributed operating system uses static taint levels for its trust system. As opposed to this method, the taint levels described for this project are dynamic, and based on real-time interactions between objects

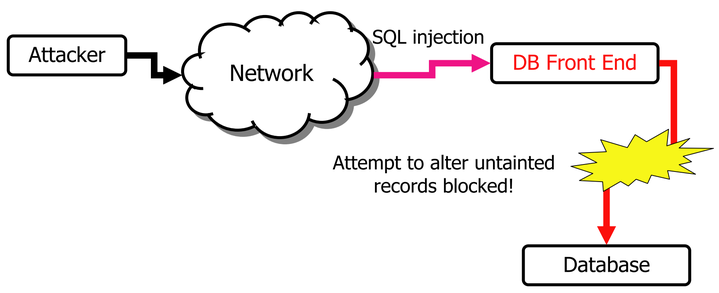
Scenario 2: Exploit to Replace Critical Files



Scenario 2: Details

1> Attacker exploits vulnerable code to remotely create spurious files; the weblog program becomes tainted when reading data from the network
2> Taint propagates from the weblog to the new files
3> Subsequently, the attacker uses the exploit to execute code that would replace legitimate files (passwords, libraries, etc) with the spurious ones
4> Attempt to overwrite untainted files with tainted ones blocked!

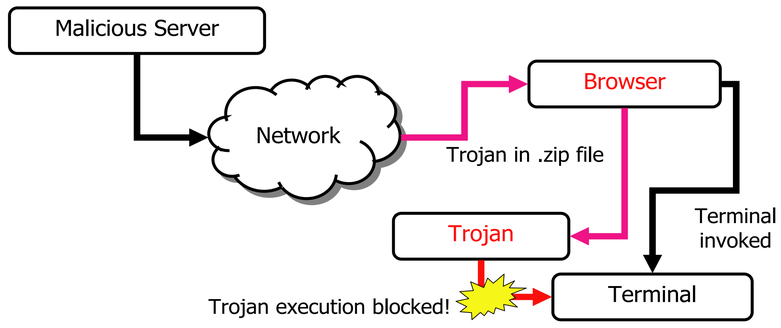
Scenario 3: SQL Injection on Read-Only Database



Scenario 3: Details

1> Attacker injects SQL commands into a database query to the front end
2> Front end becomes tainted from network input
3> Injected SQL commands attempt to overwrite critical sectors of database, but are blocked due to untainted status of this portion of the database!

Scenario 4: Activating a Trojan via a "Safe" .zip Browser Download



Scenario 4: Details

1> User configures browser to allow "safe" files to execute upon successful download
2> Malicious user serves disguised Trojan in a .zip file
3> Taint propagates to the .zip file when it is created locally by browser
4> Browser invokes a terminal which interprets a script in the archive
5> Terminal attempts to execute the Trojan, but execution is blocked due to Trojan's tainted status!

Funded by NSF REU award 0420906-CNS

PURDUE UNIVERSITY
PURDUE HOMELAND SECURITY INSTITUTE
CERIAS
REGENSTRIEF CENTER FOR HEALTHCARE ENGINEERING
Discovery Park e-Enterprise Center