Application Intrusion Prevention Systems: A New Approach to Protecting Your Data

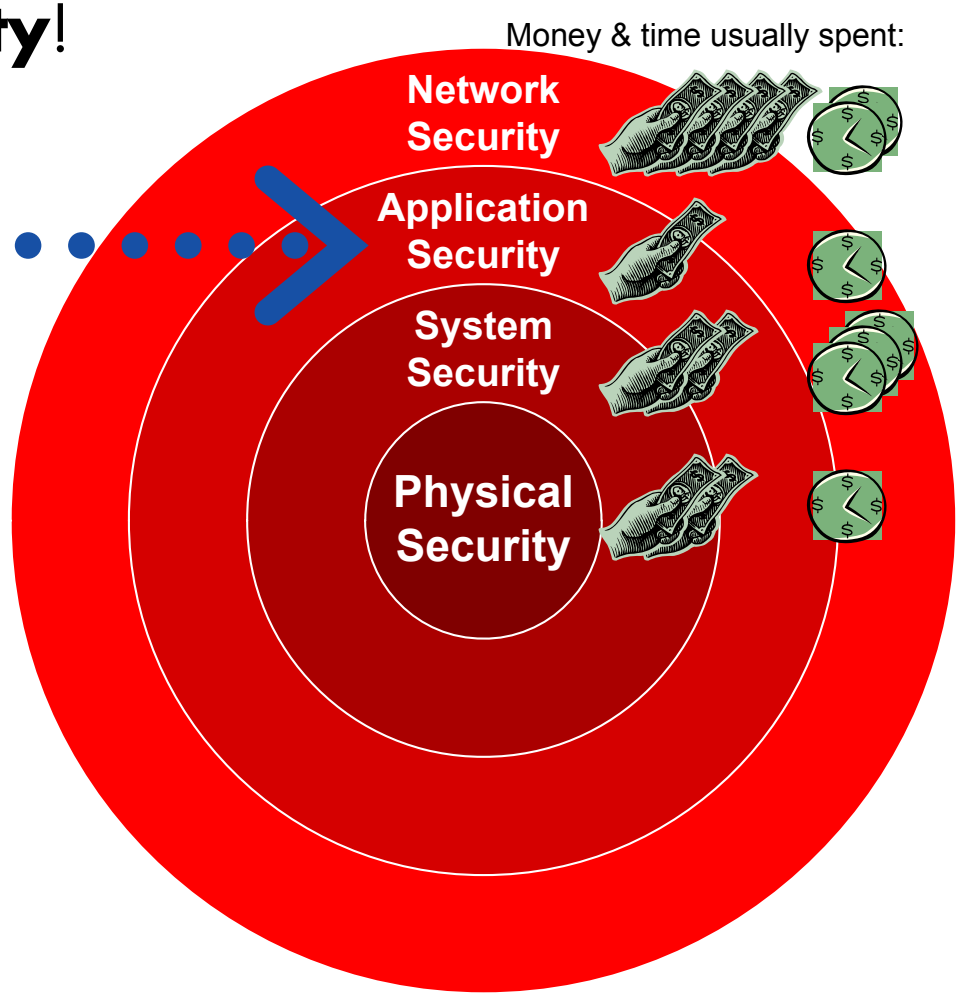
FMA-RMS
Fabrice A. Marie – 方政信
fabrice.marie@fma-rms.com

HITB SecConf2006 - Malaysia
September 18th - 21st 2006 :: Kuala Lumpur, Malaysia
DEEP KNOWLEDGE SECURITY CONFERENCE

Table of Contents

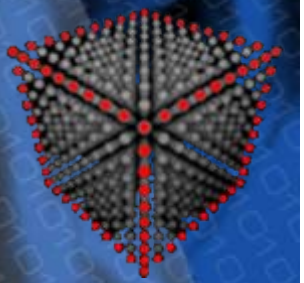
Importance of Application Security

- The most important is definitely **Physical Security**!
- Application Security should only come 3rd in your priorities



Why Hack Web Applications?

- Short answer: BECAUSE IT CAN BE DONE!
- Applications don't get the attention they deserve…
  - Why do you need a network?
  - Why do you need computers?
    - … to run applications!
- Applications attacks are much more efficient
  - Network attacks are slow and painful. Attacker needs to:
    - break 2-3 layers of firewalls
    - penetrate the system
    - escalate his privileges on the system
    - find the way to perform the fraud
  - An application attack is **faster**:
    - no need to penetrate firewalls
    - no need to penetrate the system
    - simply brutalize the application!



Why Hack Web Applications?

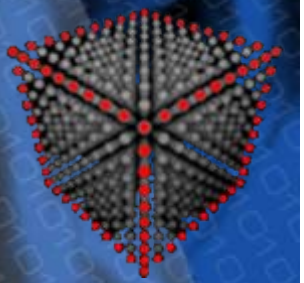
(cont'd)

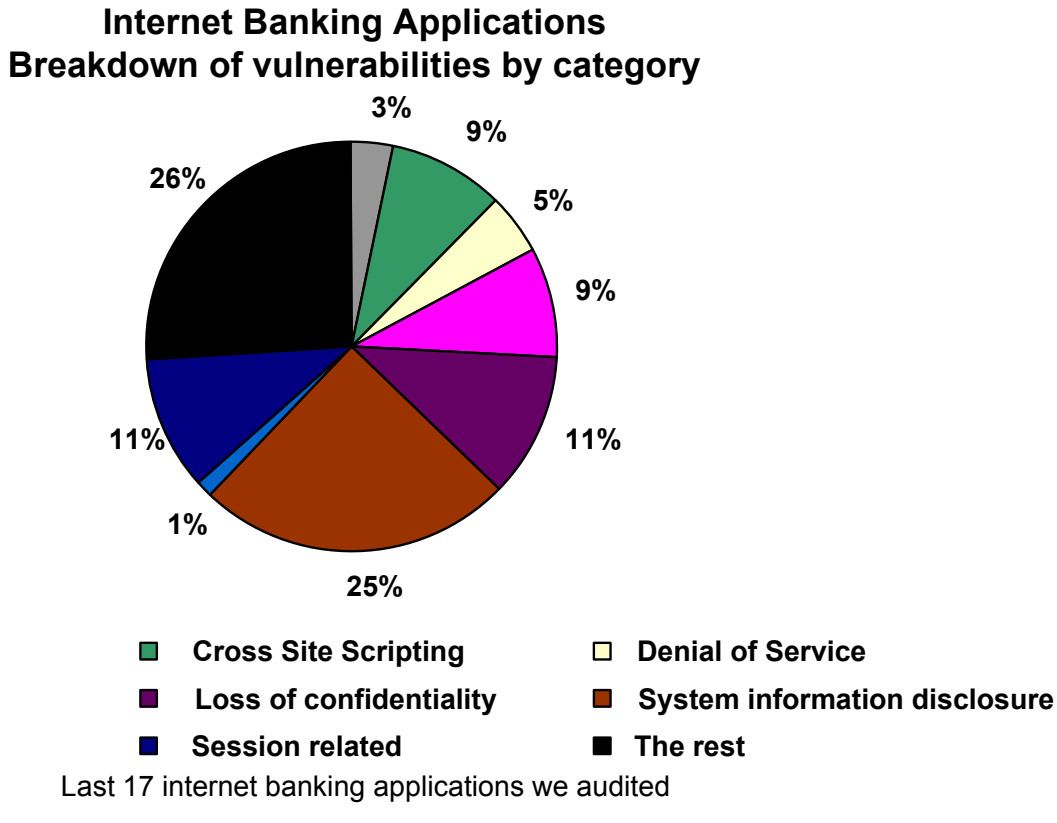
- Application attacks are generally simple
  - If not simple, then the network equivalent attack would be worse!
- Lack of skills in the application arena
  - Developers/Architects/Programmers are **under-skilled**
- You have control over your network, but not over your app
  - Network uses standard components
  - Application is a monolithic peace of software
- Because it's fun to find problems in other's work
- Because you can benefit from it!
  (and crime follows money)

Hacking Internet Banking Applications for Profit

- Frauds we commonly find on internet banking applications: Very very long list…
  - read other customer's bill payments
  - read other customer's personal information
    - very useful as the base for more advanced attacks
      - identity theft
  - read other customer's banking messages
  - stealing money using various transfer functionalities
    - direct bank transfers among others
  - buy shares at a discounted price
  - avoid transaction fees
  - various payment gateway systems replay attacks
  - destruction of transaction records
  - modification of other customer personal details
    - very useful as the base for more advanced attacks
      - user impersonation

Hacking Internet Banking Applications for Profit (cont'd)

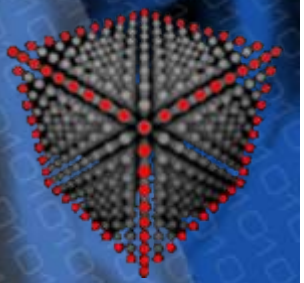




Last 17 internet banking applications we audited

Applications we could steal money from: 100%	Applications we could steal personal information from: 100%	275 vulnerabilities 429 beta scripts 341 unnecessary files

average: **16 vulnerabilities** per application



Why Bother With Detection?

- A good application wouldn't require detection
  - the attacker simply would not get through
  - If an attacker cannot get through why bother detecting?
    - eg: lots of firewall rules are not logging to avoid noise
- Statistically, there is almost no good web application when it comes to security
  - ratio good applications vs. bad applications is tragically unbalanced
- The only goal of detection of application attack is **prevention**
  - lock the account of the offender
  - sue the offender if there is substantial proof
  - other **actions**

Why Bother With Detection? (cont'd)

- Why prosecuting offenders before they even succeed?
  - Very few people prosecute network reckons
    - due to the simplicity/complexity of TCP/IP protocols
      - port mapping, ping sweeps, ARP mapping, and more artillery
      - sometimes impossible to differentiate from normal usage

    → proof is hard to behold in court
  - Application reckons on the other hand leave hard proof
    - tampered data flow can be detected
    - definitely intentional and can be proved to be as such.

    → proof **can** behold in court

- Right now secure applications stop attacks (very few)
  - Using strict validation
  - Using strict logic control, and flow control
  - But they only treat these as **mistakes** instead of **attacks**
    - they do not prevent further attack: **no ACTION**

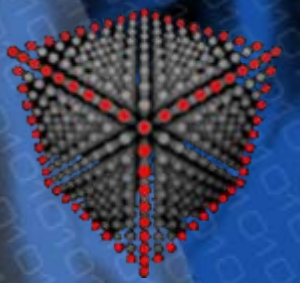
Problems of Modern IDS Technologies

- Almost the only technology that flag an attack as an attack

- Intrusion must first be defined before it can detected…
  - Classic network intrusion leave traces and symptoms that network IDSes can detect
    - reverse root shell, suspects string in protocols, other anomalies
  - Classic intrusion leaves traces and symptoms that host IDSes can detect
    - modified files, suspect log entries, other anomalies

- **How** do you define an application intrusion/abuse?

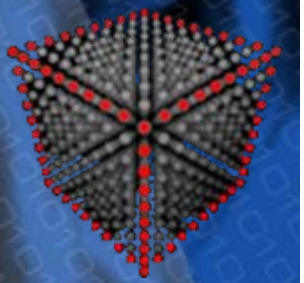
Problems of Modern IDS Technologies (cont'd)



- How do you define an application intrusion/abuse?
  - Same tactics can be used to detect classic attacks
    - SQL injections
    - XSS attacks
    - username/password brute-force
    - buffer overflows

- Just how can the IDS understand a logic flaw ?
  - e.g.: IDS has no knowledge of bank account numbers
    → It would not know that I transfer money from a victim's account instead of from my own account

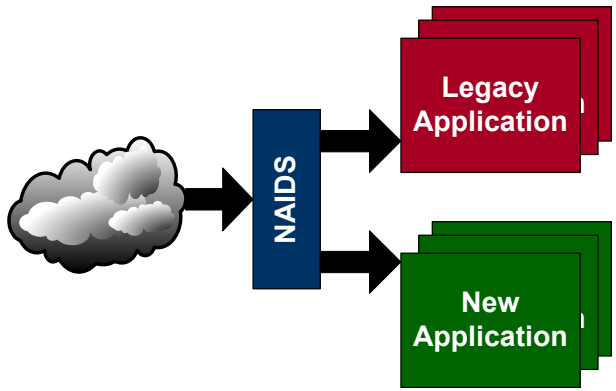
Proposed IDS Technologies



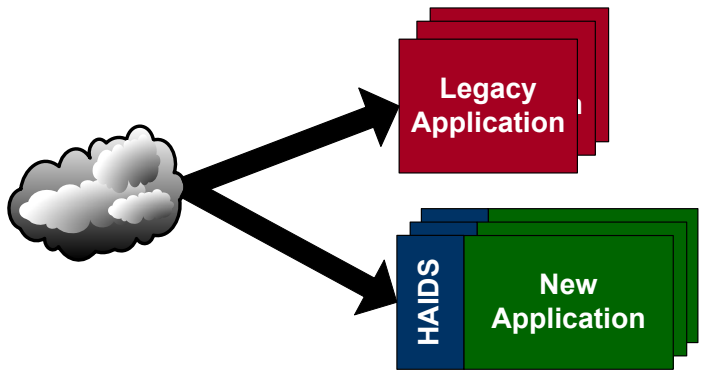
- Network-based Application Intrusion Detection Systems (NAIDS)
  - have to be generic to monitor any web application
  - and as such can only detect generic attacks
    - SQL injections, XSS, buffer overflows, brute-force, etc…
- Host-based Application Intrusion Detection System (HAIDS)
  - built into the application using a special framework
  - has a complete understanding of the application, its parameters, and its business logic
  - knows what is merely a mistake and what is a blatant attack

Proposed IDS Technologies (cont'd)

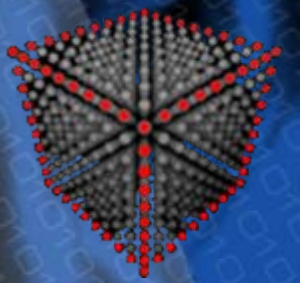
- NAIDS
  - Is an advanced generic filtering HTTP proxy



- HAIDS
  - Is an advanced framework on which the application is built



Proposed IDS Technologies (cont'd)

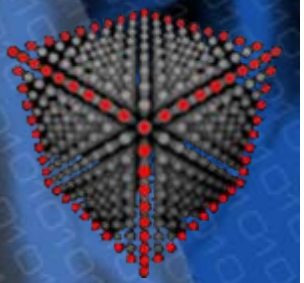


Main goal:

- Differentiate between an **ATTACK**
- And an **ANOMALY** or normal usage

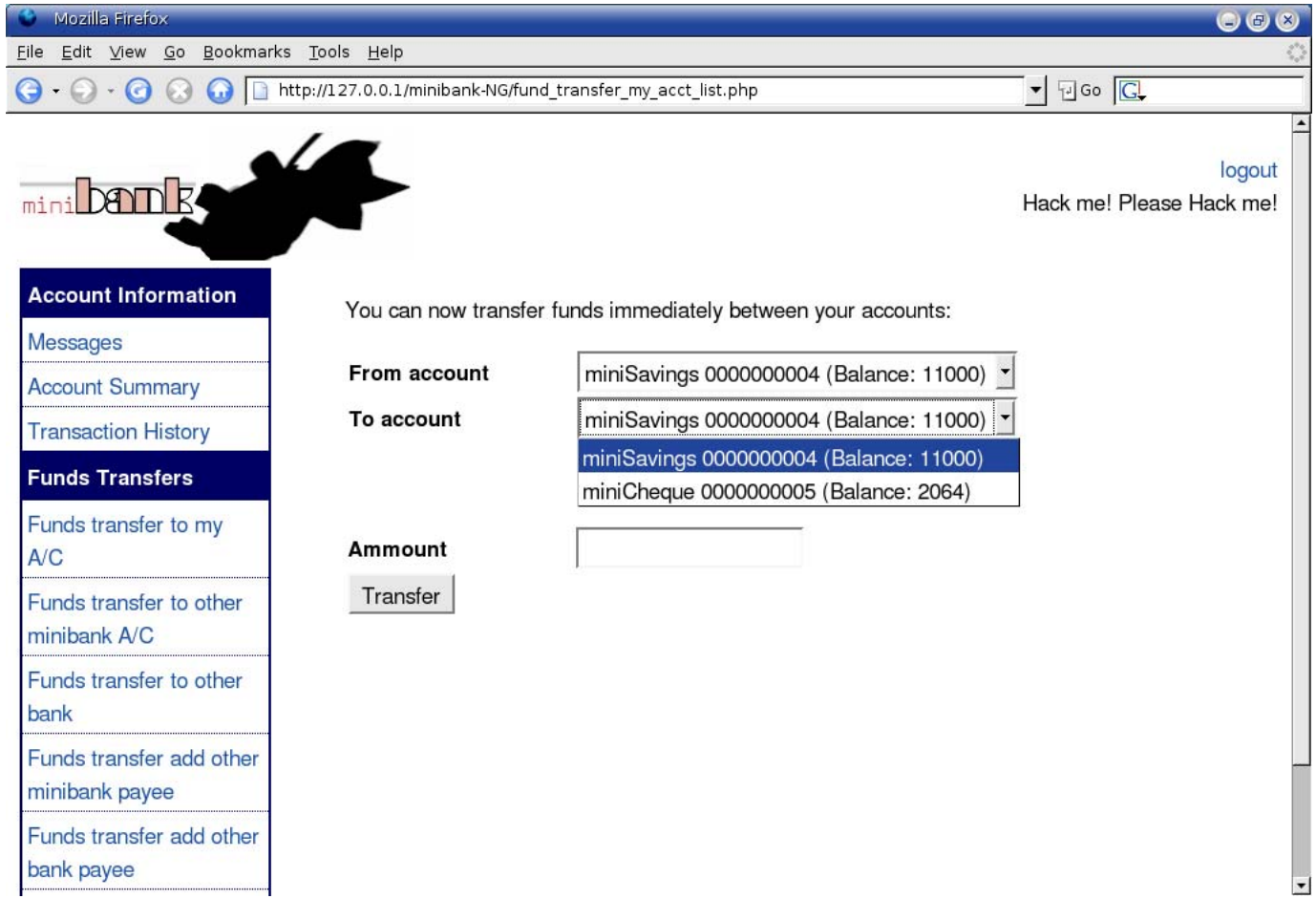
- Most of this presentation lists various classic attack patterns that
  - Black/grey/blue/white hats use when they attack apps
  - Are fool proof
  - Have very few false positive

Generic Tampering Detection

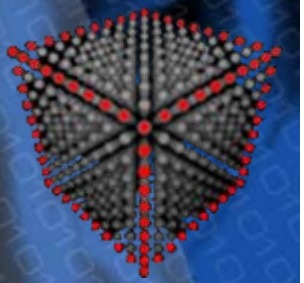


- Web applications use dialogs to interact with the user:
  - radio buttons and check-boxes
  - fields
  - hidden fields
  - drop-down lists
  - select list
  - and more widgets…
- Some are free-form
  - e.g.: user can enter freely text
- Some are limited/restricted (supposedly)
  - e.g.: drop-down lists limit the user's choice

Generic Tampering Detection (cont'd)



- "to account" is a restricted parameter
- "amount" is free-form field

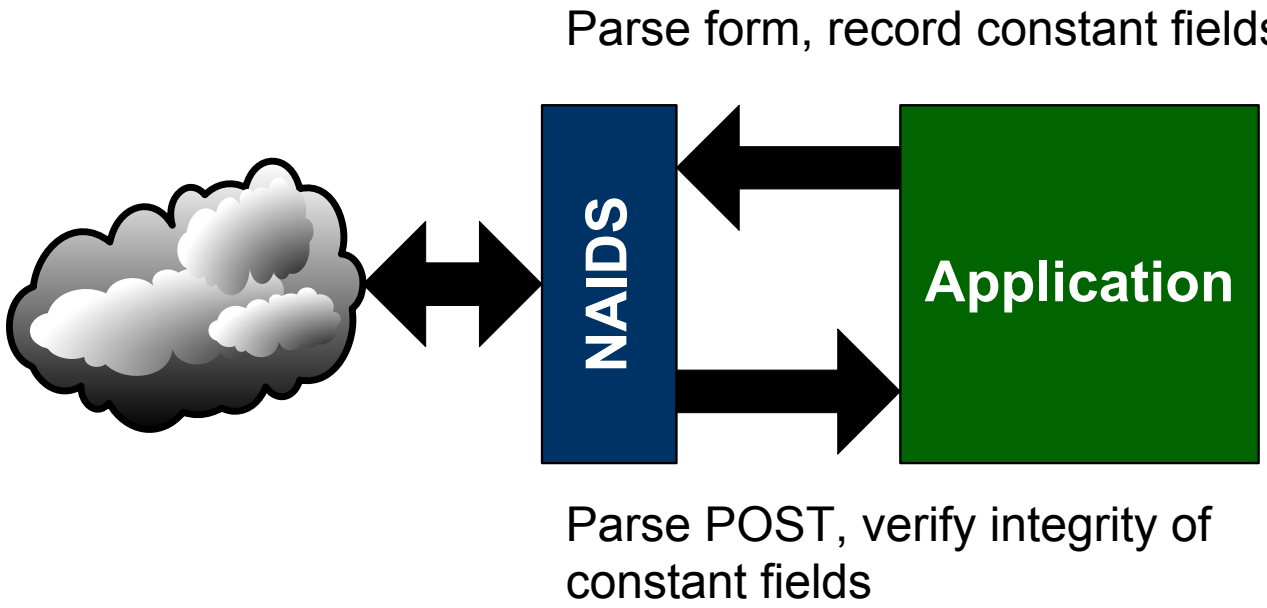


Generic Tampering Detection (cont'd)

- Restricted parameters cannot be changed by users (supposedly)
  - drop-down lists (**<select><option>**…)
  - radio buttons (**<input type="radio"** …)
  - check-boxes (**<input type="checkbox"** …)
  - hidden fields (**<input type="hidden"** …)
  - fixed length regular text fields (**<input maxlength="10"** …)
  - cookies
- So only attackers would modify them (using proxies)
  - If you changed such parameters you had an **agenda**
  - The server side set these parameters before sending them to the client
    - The server side therefore can verify and detect modifications easily

Generic Tampering Detection (cont'd)

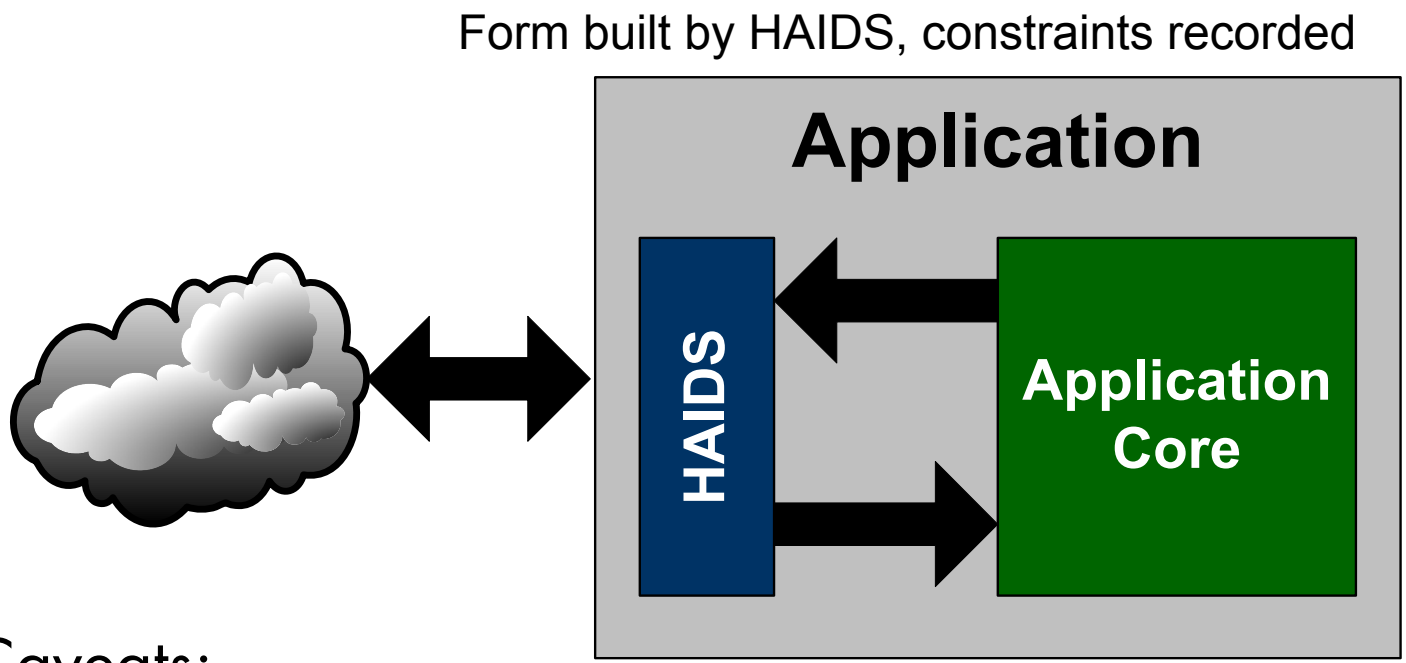
- Can be implemented on an NAIDS



- Caveats:
  - Bad HTML? (classic)
    - Parsing errors
  - No HTML/form ? (XML-RPC, SOAP, AJAX, etc…)
    - Can't record constant fields, so can't check them later

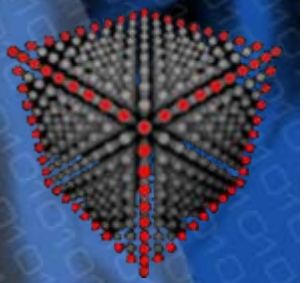
Generic Tampering Detection (cont'd)

- Can be implemented on an HAIDS



- Caveats:
  - Only works for apps that were built with the framework

Attack Behavior Detection



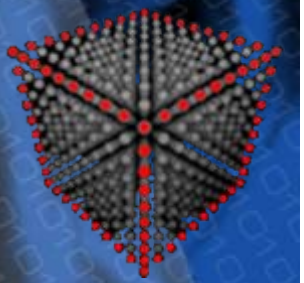
- Regular users do not make repetitive mistakes
  - some repeated mistakes are obvious attacks
  - and should trigger alerts and/or action

- Mistakes that are not mistakes:
  - Authentication mistakes:
    - e.g.: failing a username / password challenge more than 5 times in 1 minute.
  - Validation error
    - e.g.: blatant SQL commands instead of an email address
    - e.g.: blatant SQL commands instead of a numerical itemID
    - e.g.: HTML/JavaScript reserved words instead of a family name
    - e.g.: blatant buffer overflow (e.g.: string longer than 200 chars)

User mistakes? my foot! **Attacks** certainly.

Attack Behavior Detection

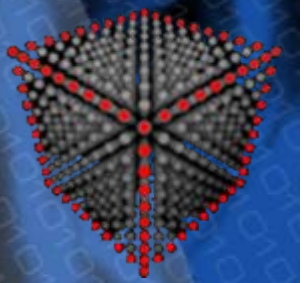
(cont'd)



- Mistakes that are not mistakes: (cont'd)
  - Ignoring the regular business/data flow
    - e.g.: going straight to the *purchase confirmation* page before having clicked on *check-out cart*
  - complex form filled too quickly by the user
    - e.g.: a form with 50 fields getting filled under a second by the user
  - blind users?
    - e.g.: forms failed 5 times in a row the CAPTCHA verification
      - either user is genuinely blind, or it is an automated attack!

Attack Behavior Detection

(cont'd)



- In NAIDS/NAIPS we can implement
  - XSS detection
    - Identifying classic XSS patterns (**<script** etc.…)
      - Require rule exceptions for some apps
  - SQL Injections
    - Identifying classic SQL Injections patterns (**' or 1='1** etc…)
      - Require rule exceptions for some apps
  - Buffer Overflows / Remote command execution attacks
    - Identifying super long strings and command execution patterns (**NOPs**, **/bin/sh**, **cmd.exe**, etc…)
- All of them have
  - Lots of false positives
    - Normal usage flagged as attacks
  - Lots of false negative
    - Attacks not flagged as attacks

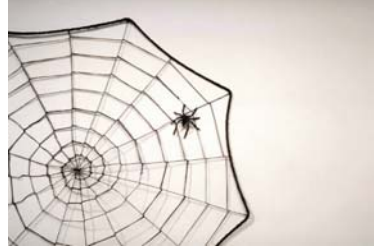
HAIDS → HAIPS

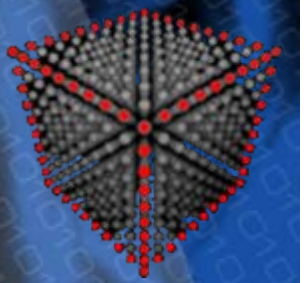
- Several goals of detecting an attack (as opposed to just stop it quietly):
  - know that your application is under attack
    - you have no idea....!!!
  - know **who** performed the attack
  - know **what** the attacker attempted
    - so you can know what seems to be weak and deserve more attention
  - **Prevent** the current attack and/or further attacks
    - lock the account automatically
    - arrest and prosecute the offender
    - possibly other actions
- To turn a Host-based Application Intrusion Detection System into a Prevention System, we need to take **actions** instead of just alerting...

HAIDS → HAIPS

- Actions that an HAIPS could implement:
  - Log the attack in details
    - send a generic non-informative error message back to the client
    - log a complete and accurate error message on the server
  - Email application administrator with full error message
  - Email security department with full error message and session
  - Send SMS
  - Lock the user account
  - Issue a challenge to deter and verify automatic attack
    - could be a CAPTCHA or any other more personal question that only the real user would know
  - Redirect to a warning page
    - *yeah right! Since when do you want to warn attackers?*
  - *Let the fun begin…*
    - redirect the user to a honeynet
    - send back garbage to request from that user the next 5 mins

HAIPS Framework

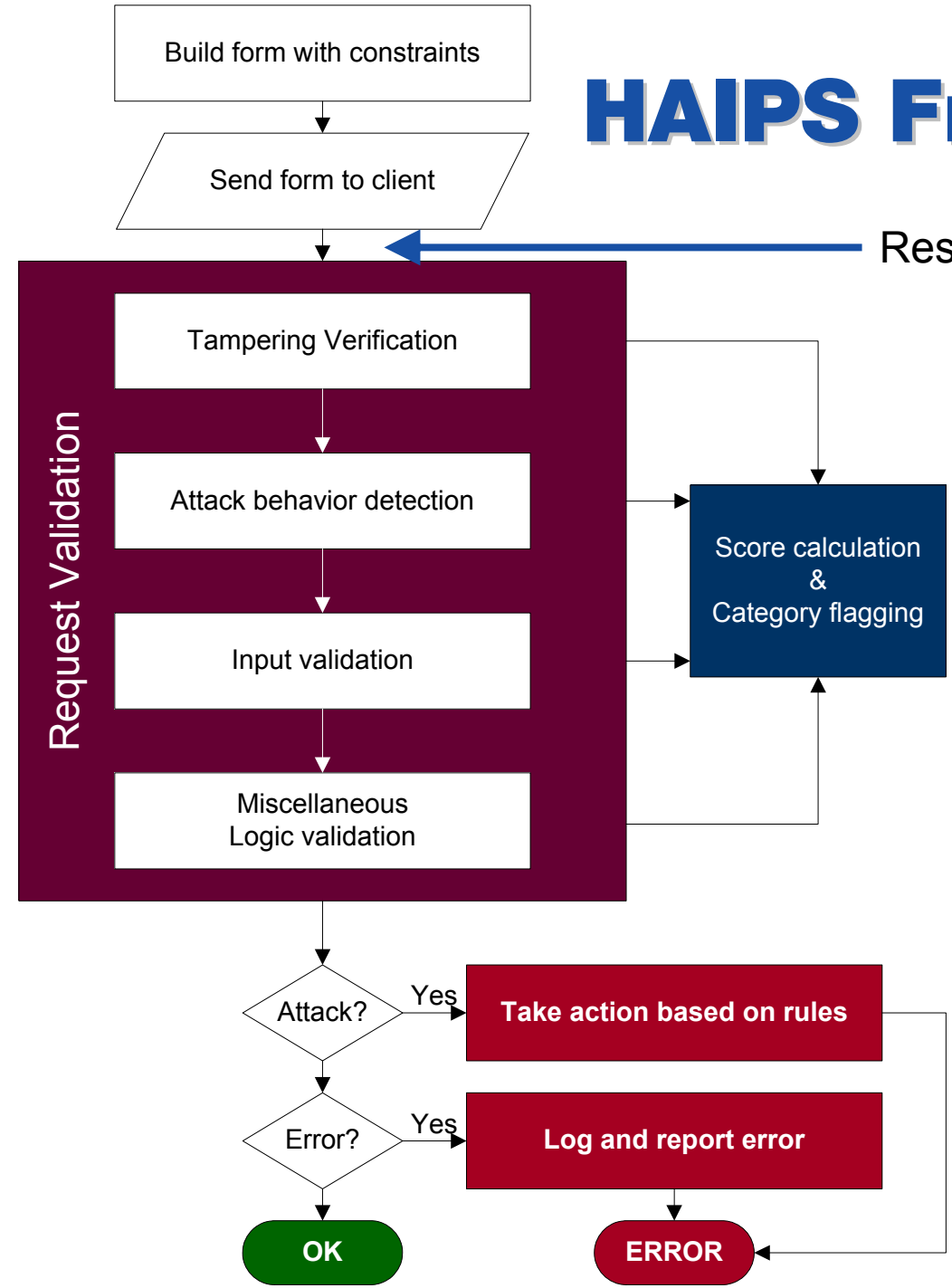




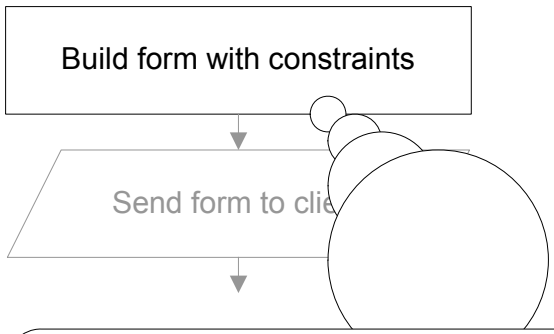
- HAIPS has to be implemented in every application you want to protect
- This is best done using a framework

- We will try to propose such a framework

HAIPS Framework (cont'd)



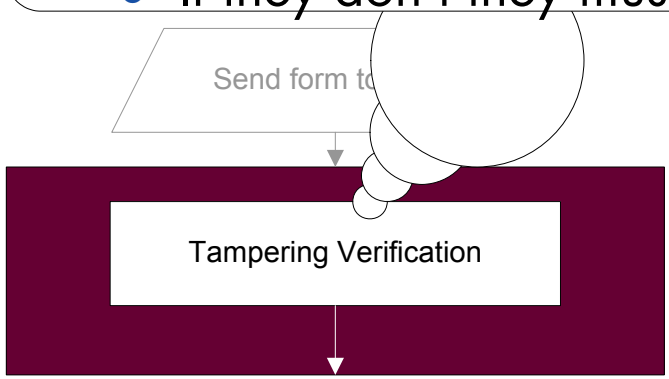
HAIPS Framework
Build Form with Constraints



- Mark hidden fields as hidden
- Mark maximum length in relevant fields
- Auto-generate client-side JavaScript validation code.
- Remember values of cookies, hidden fields, and values that should not be tampered with
  - So we can verify them later…

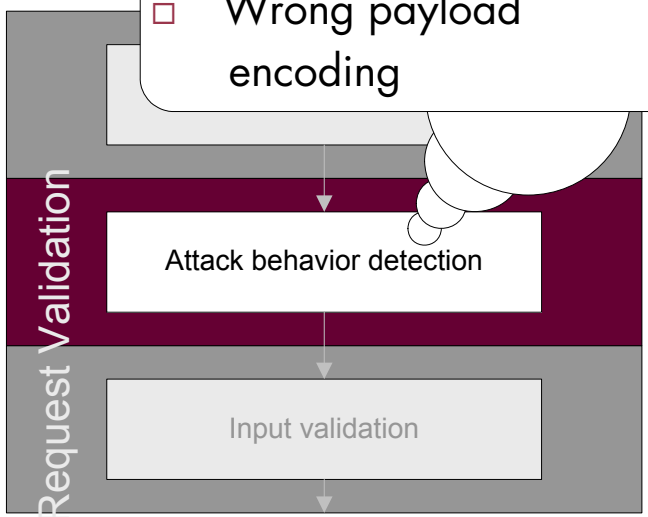
HAIPS Framework Tampering Verification

- Verify that immutable fields have not been tampered with:
  - drop-down lists
  - radio buttons
  - check-boxes
  - hidden fields
  - fixed length regular text fields
  - cookies
- Verify that the parameters indeed exist
  - If they don't they must have been removed…



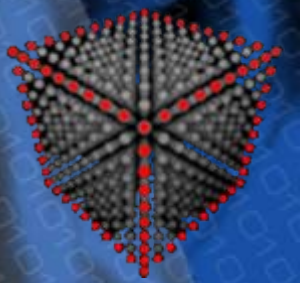
HAIPS Framework
Attack Behavior Detection

- Consecutive errors
- SQL injections
- Cross Site Scripting
- Buffer Overflows
- Missing cookie
- Missing or invalid referrer
- Modification of user-agent mid-session
- missing parameter
- Wrong action GET/POST
- Wrong payload encoding
- Wrong header encoding
- Suspect URL
- booby-trap triggered
- other classic injections
- additional parameters not supposed to be there
- Role bypass attempt
- Other bypass of client-side validation



HAIPS Framework Attack Behavior Detection

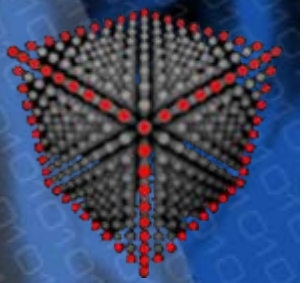
(cont'd)



- Consecutive errors
  - e.g.: 5 failed login attempts in 1 minute
- SQL Injection
  - e.g.: date containing a ' or other SQL reserved characters
- Cross Site Scripting
  - e.g.: name containing <script> or other typical XSS thing
- Buffer overflow
  - e.g.: parameter more than twice longer than expected
- Missing cookies
  - e.g.: themeID cookie in a forum, missing
- Missing referrer
  - e.g.: user/attacker is using a proxy that filters it out or set the browser to ignore them. Punish the user!

HAIPS Framework Attack Behavior Detection

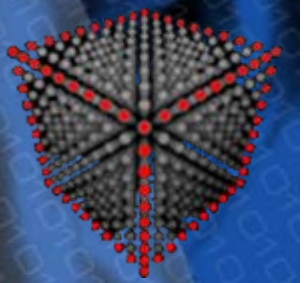
(cont'd)



- Missing parameter
  - e.g.: one of the mandatory parameters is missing, and JavaScript should have prevented the user from submitting the form
- Modification of user-agent in mid-session
  - e.g.: the user logged-on the application using Firefox but subsequently the browser advertise itself as IE…
- Invalid Action
  - e.g.: the request was supposed to be POSTed but instead it gets GETed or vice-versa.
- Wrong payload encoding
  - e.g.: form was supposed to be in application/x-www-form-urlencoded but instead get posted in multipart/form-data or vice-versa.

HAIPS Framework Attack Behavior Detection

(cont'd)



- Wrong header encoding
  - e.g.: the attacker did not URL encode properly his request…
- Suspect URLs
  - e.g.: URLs containing parameters that contain a leading / or ../
  - e.g.: URL containing reserved filenames
    - web.config
    - WEB-INF
    - .bak
    - blahblahblah~ (Unix-style backup file)

HAIPS Framework
Attack Behavior Detection

(cont'd)

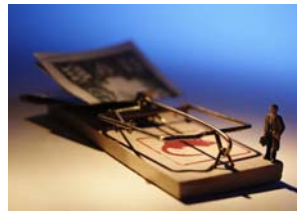
- Booby-trap triggered
  - "must .... press .... red .... button.... !!" or
  - "I wonder what this is for?"
  - e.g.:


```
<form action="login.jsp" method="post">
<input name="username" maxlength="10" />
<input name="password" type="password"
       maxlength="100" />
<input type="submit" value="Login" />
<!-- <input type="hidden"
            name="is_admin"
            value="0" /> -->
</form>
```

- Typically the attacker **will** have itchy-hands
  - and will uncomment the above and set **is_admin** to 1
    - uncommenting it obviously triggers our alarm/trap...

HAIPS Framework Attack Behavior Detection

(cont'd)



- Booby-trap triggered
  - More examples:
    - User attempts to log-on as 'admin' → 'admin'
      - … and you made sure the admin user is called differently
    - Authenticated normal user tries to access **`/admin`**
      - … and you made sure that admin area is called **`/a`**
      - … and the guy is not even an admin!
- Who has never tried his luck with these during an assessment ??

HAIPS Framework Attack Behavior Detection

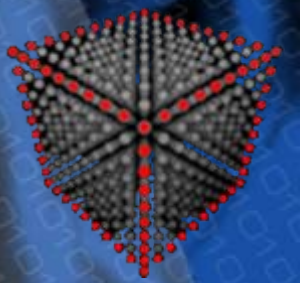
(cont'd)

- Other classic injections
  - DAP/LDAP injection e.g.: a family name contains no *or ,
  - CR/LF injection e.g.: a family name contains no CR or LF.
  - Shell command injection e.g.: a username contains no ;
  - XPath injection e.g.: a username contains no ' or =
  - Cobol field injection… nah just kidding :-)

- Additional parameters not supposed to be there
  - Sometimes attacker try their luck (you'd be surprised how often it works…) by adding undocumented and unexpected parameters
    - e.g.: **`is_admin=1`**
    - e.g.: **`loggedon=true`**
    - …

HAIPS Framework Attack Behavior Detection

(cont'd)



- Role bypass attempt
  - e.g.: a user logged-on as regular user who try to enter directly the admin command screen URL when it does not even appear in his menu
- Any other client side validation bypass
  - if a user bypasses whatever trivial JavaScript validation it means he is attacking!
  - We cover most of them earlier things
- Feel free to add your own methods of discerning between a hand-made request and a user-clicked request in the browser…

HAIPS Framework Input Validation

Validates all your data-types

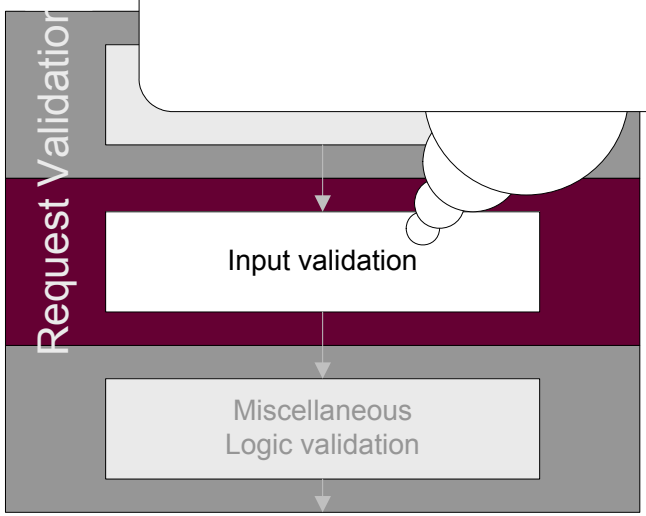
- Dates
- Zip codes
- Phone numbers
- Addresses
- Names
- Amounts
- Email addresses
- Usernames
- etc....

Some can count towards intrusion detection:

- dates if selected from a JavaScript calendar cannot be wrong
- if wrong it means attack...

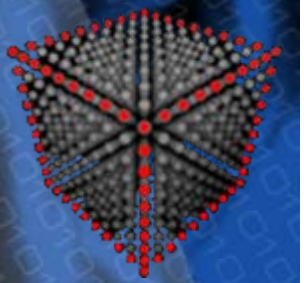
Some cannot:

- e.g.: no way to know if the name *thisisabadname* is indeed a real name or not



HAIPS Framework Attack Behavior Detection

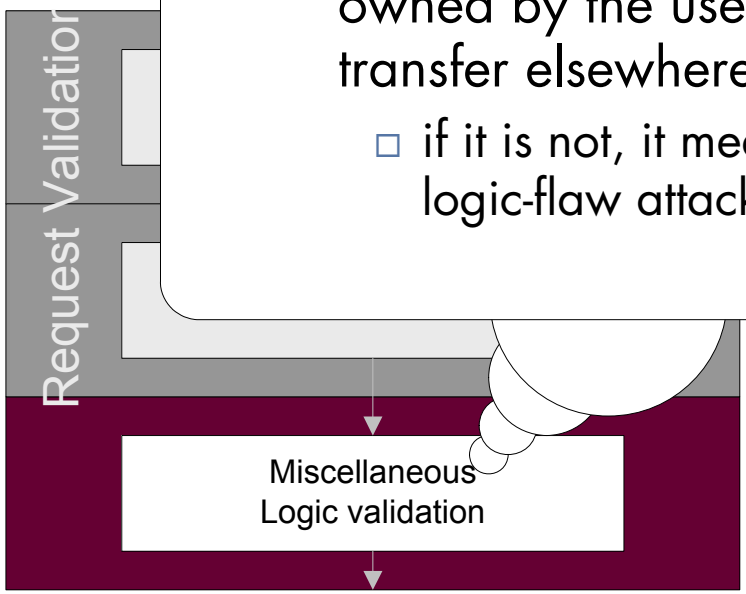
(cont'd)



- Many different types of data to validate
  - user must provide a call-back for each type
  - the framework maker could pre-write some of the classic types
  - The user simply would have to add the missing ones he needs
  - use of Object Oriented Language and inheritance makes the tasks much easier and cleaner.

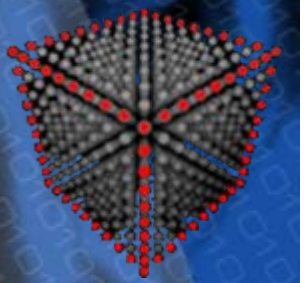
HAIPS Framework
Miscellaneous Logic Validation

- This is were you detect and protect against logic flaws
  - e.g.: in internet banking, check that the account is owned by the user before sending back the details
    - if it is not, it means the user tried to perform a read logic-flaw attack
  - e.g.: in internet banking, check that the account is owned by the user before taking money from it to transfer elsewhere
    - if it is not, it means the user tried to perform a write logic-flaw attack



HAIPS Framework Attack Behavior Detection

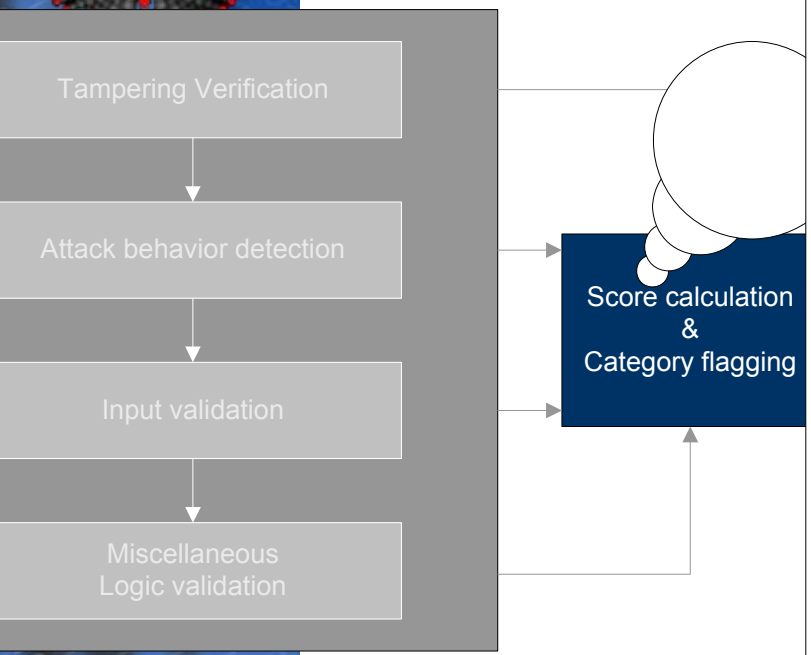
(cont'd)



- Logic flaws depend of the business logic of the application
- The user will have to provide call-backs that will do the verification

- Again, the extensive use of Object Oriented Languages and inheritance will make the task simpler

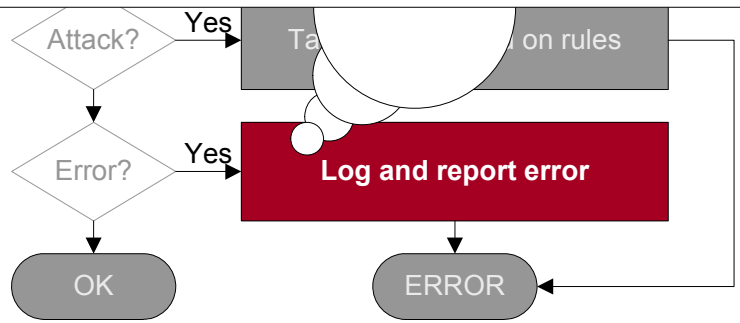
HAIPS Framework
Score Calculation & Category

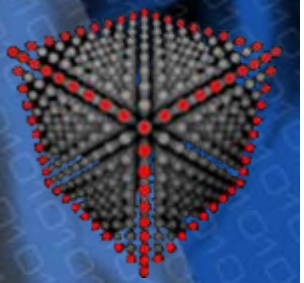


- Scoring:
  - Each negative aspect previously mentioned add to the attack-score of a request
  - Nastier attacks get bigger scores
  - Allows the application owners to set thresholds for alert and thresholds for preventive actions
  - False alarms can be avoided by using negative attack points to work around known browser bugs
- Category flagging:
  - Is simply the process of deciding if there an error due to "normal" conditions or if it is indeed an attack

HAIPS Framework
Take Action Based on Rules

- Log and report error is very straight forward
  - Log a very detailed error message to a file or database containing all the info possible:
    - e.g.:
    - time/date
    - username, ip address
    - cause: e.g.: **`SQL injection on username=x‘ or 1=‘1`**
  - Send back a generic error message to the client
    - e.g.:
    - **`Service Unavailable. Try again later.`**





Caveats Drawbacks Problems

Just like every security frameworks, it is not perfect

- The **main problem** of this framework is obviously **the developer that uses it** !!!!!
- The developer that uses it have to
  - understand the reason behind the framework
  - understand how his application could get attacked
  - in order to
    - protect it in the first place
    - put in some detective controls using this framework
    - put in some preventive/corrective controls using this framework
- The framework helps the developer, but it cannot replace a good brain with common-sense…

Caveats Drawbacks Problems (cont'd)

- The framework can only protect an application that was written with it
  - It will not auto-magically support your legacy application
- The framework can only protect applications that are written half-properly or better
  - Some applications violates their own rules so the framework would flag unusual activity as attacks, wrongly
- What about Flash forms ?
  - They could be supported,
    - additional work to tell the framework
      - about the form content
      - the constant field values
      - and various other parameters like content-encoding and request method

Caveats Drawbacks Problems (cont'd)

- The major technical drawback (as it is) of this method is remoting technologies:
  - Java / Javascript / Flash / ActiveX with
    - AJAX
      - http://en.wikipedia.org/wiki/AJAX
    - JSON-RPC
      - http://json-rpc.org/
      - http://oss.metaparadigm.com/jsonrpc/
    - XML-RPC
    - Corba
    - Direct sockets with esoteric or proprietary communication protocols
  - The framework has to be built for it in mind
    - it would be easier to write a new framework using the same ideas
      - to cater for XML buffers instead of HTML widgets in one direction and XML buffers in the other
      - Lots of additional checking to perform

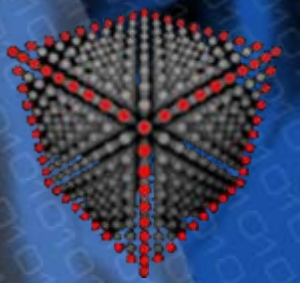
Caveats Drawbacks Problems (cont'd)

- The framework must integrate with standard frameworks
- Java:
  - Struts, Java Server Faces, Tapestry, OWASP Stinger, etc…
    - Should our framework integrate with them all or only one? Which one?
    - Integrate HIPS with these, or integrate these with HIPS?
- .Net:
  - Built-in .Net Validator mechanism
- Every application platform would need its own integration..

Caveats Drawbacks Problems (cont'd)

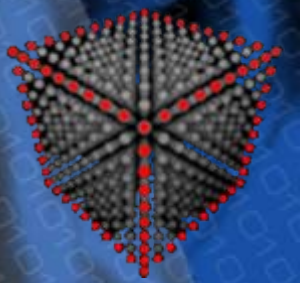
- False alarms
  - Every IDS has false alarms
  - This one potentially too
  - They are almost inexistent though:
    - Because we know really well what we expect
      - No such thing as an 'accidental SQL injection'…
    - Proper tuning of the scoring system is an advantage
      - Missing cookie can be a small 'offense'
        - Sometimes missing genuinely
      - Booby trap triggered are obviously a 'death sentence'
        - No coincidence there
      - Detected logic flaw attack is also a 'death sentence'
        - Bank account numbers just don't get changed by mistake

Caveats Drawbacks Problems (cont'd)



- The final problem is
  - We haven't implemented this framework yet…

Any volunteers ?



Conclusion

- At the moment security layers controls are
  - preventive controls
    - Network: firewalls, NIPS
    - System: HIPS
    - Application: input validation frameworks
  - detective controls
    - Network: NIDS
    - System: HIDS
    - Application: none, or manual using the log files
  - corrective controls
    - Network: NIPS
    - System: HIPS
    - Application: none, or manual

- There is a need for HAIDS, HAIPS, NAIDS, NAIPS !
  - Don't be shy. Any volunteer again?

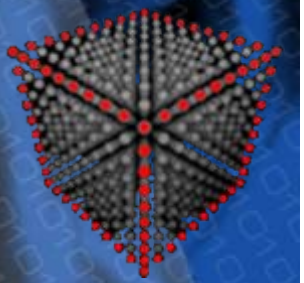
Conclusion

(cont'd)

- Web application security is still very young
  - technologies take time to be invented
  - technologies take time to mature
  - products and offering take time to become robust
- Method proposed
  - is relatively simple
  - straight forward
  - relatively low false positives and low false negatives
  - Not easy to integrate cleanly with existing frameworks
- In a nutshell it's not there yet
- and it will take some time to be robust!

Conclusion

(cont'd)



Bad:

- HAIPS will be the worse nightmare for app tester
- Even worse for automated application assessment tools!!!

Good:

- natural selection of security consultants

Questions ??

FMA-RMS
Fabrice A. Marie – 方政信
fabrice.marie@fma-rms.com