Taintless

Defeating taint-powered protection techniques

Abbas Naderi (aka AbiusX)
Mandana Bagheri
Shahin Ramezany

Covered Topics

Before We Begin

While you obtain the tools and get ready, we'll warm-up our systems.

Getting To Know Taint

What is Taint? What types of taint are there? What processes use taint to defeat cyber-attacks?

Existing Techniques

Studying a select group of candidate taint-based techniques helps us better understand -and hence defeat- taint.

Taintless

Describing the tool, its modes of operations and goals.

Demonstration

Trying Taintless on a bunch of software, attempting to analyze and bypass their protections and weaknesses.

Q&A

Covering any final thoughts the audience might have.

“if it breaks you, it makes you stronger”

Before We Begin

Let's warm-up our systems by solving this challenge while you get the tool:

```php
<?php
$in=$_GET['q'];
if (preg_match("!^\d+$!",$in) && md5("abiusx.com 1853")==$in+"x")
    print("Key is: ".substr(md5($in),-6));
else
    "Nada.";
```

You can't run code on your brain! (Or can you?)
http://ideone.com/C7bOrg

github.com/abiusx/taintless (needs composer)
github.com/abiusx/WP-SQLI-LAB
github.com/abiusx/WP-SQL-SINK

If you solved both challenges, find harder ones on my twitter.com/abiusx

“Data! Data! Data!" he cried impatiently. "I can't make bricks without clay.”

What is Taint?

Sources of Taint

- Just like in real life, sources of taint are typically people
- Applications are designed to work well with proper input
- Improper input makes a program sick
- Sick programs behave differently and unexpectedly

What is Taint?

Tainted Input

- User-input to an application is generally considered tainted
- Specially on web, were anyone can visit!
- Tainted input needs to be sanitized before use in the application
- Everybody knows that, nobody does that.
- Our forefathers didn't even know that (Legacy Code)

What is Taint?

Sinks

- Everything entering the application system is categorized as tainted (e.g Second order attacks)
- Taint propagates throughout the program, until it reaches a sink
- A sink is a [security] critical operation inside the application (e.g database query)
- Sinks are important, just like body organs, as tainted input aims that specific organ.
- Sinks are wrapped in taint-based techniques

What is Taint?

Taint Propagation

- The more complex a code-base, the more possible means of taint spreading around
- Just like a virus in our body, taint can play hide and seek to bypass all sentinels and filters
- Taint may totally change form, typically rendering it harmless, but sometimes this change morphs it into something dangerous (e.g encrypting an innocent string into a piece of code)

Taint Tracking

What is Taint Tracking?

- Traditional taint-based technique for protecting applications is known taint tracking
- Already available in core at Perl, Ruby, PHP and many others as extensions
- Intensive processing, impossible to accurately model
- Typically performed on strings, treating them (or individual characters) as black and white (and sometimes gray)
- String operations throughout the program propagate the taint
- Taint is increased, reduced or morphed in the process

Taint Tracking Example 1

```
<?php
$x=$_GET['input'];
$y=substr($x,0,10);                    //reduced

$z=str_replace($x,"a","b");            //modified

$w=str_repeat($x,3);                   //increased

mysql_query_("SELECT * FROM users WHERE username='{$y}'");
```

Taint Tracking Example 2

```
<?php
$x=$_GET['input'];


if ($x*1>0) //its a number


mysql_query_("SELECT * FROM users WHERE userid={$x}");
```

Taint Tracking

Sink Analysis

- Parses SQL query (or any other expected data) and marks critical (security-intensive) tokens
- If taint exists in (or conforms) these tokens, disinfects
- Easiest disinfectant is exit(-1)
- Policies define what to do with gray areas.

Gray Taint

- If an string operation fades tainted data into mixed data, disallowing a one-to-one mapping (or modeling), gray taint is made

- Example:
  $x=$_GET[‘input’];
  $y=preg_replace($x,”(\d).(\d)”,”9$29$19”);
  $z=md5(“username=‘{$x}’”);

- $y has gray taint because it’s hard to model regular expression taint propagation

Gray Taint (2)

- Example:
  $x=$_GET[‘input’];
  $y=preg_replace($x,”(\d).(\d)”,”9$29$19”);
  $z=md5(“username=‘{$x}’”);

- $z has gray taint because its impossible (infeasible) to model md5 taint propagation

- It’s not always impossible for the attacker!

Treating Gray Taint

- Whether to consider gray taint as safe or unsafe, is a matter of threshold.
- Thresholds result in false negative and positives
- Most solutions claim to handle gray taint well, but non of them actually do. They just ignore it to make the program work, rather than stop them and break the code.
- Totally in contrast with what our bodies do!

Positive Taint

- So far all taint mentioned was negative taint, i.e bad
- Positive taint is what we know to be good:
  - Track it and assume everything else to be bad (just like our bodies)
- Will break the programs more, but is intrinsic to the nature of application (no attacker control)

Positive Taint Tracking

- Very few solutions for positive taint tracking
  - e.g Diglossia, Halfond et. al.
- They suffer from the same propagation hardships of negative taint tracking
- Hard to model many operations
- Impossible to model some others
- Typically configured very loosely

“The world is full of obvious things which nobody by any chance ever observes.”

Inferring Taint

- Since we can't track taint accurately, and are bound to approximation; why not employ approximation from the start?
- Instead of tracking taint from application input to the sink, modeling every organ in its complicated body; inspect the value from time to time, and infer which parts are tainted

-

- Way lower accuracy, way more simple and fast

Taint Inference

Example

- ```
  <?php
  function mysql_query_($query) {
    $input=$_GET['u'];
    $len=strlen($input);
    $match=substr($query, strpos($input,$query),len);
    if (levenshtein($match,$input)/$len<0.1) exit(-1);
  }
  mysql_query_("SELECT * FROM users WHERE username='{$_GET['u']}' ");
  ```
  ```

Feasibility

- Approximating input/output correspondence seems very easy, but is actually very computation hungry

```
foreach $query in $queries
  foreach $input in $inputs
        $match=approximateFind($input,$query);
        $distance=stringDistance($match,$input) /
  length($match)
        if ($distance>$threshold) die();
```

O(x L x M x I)
N=number of queries, M=number of inputs, L= query size, I = input size

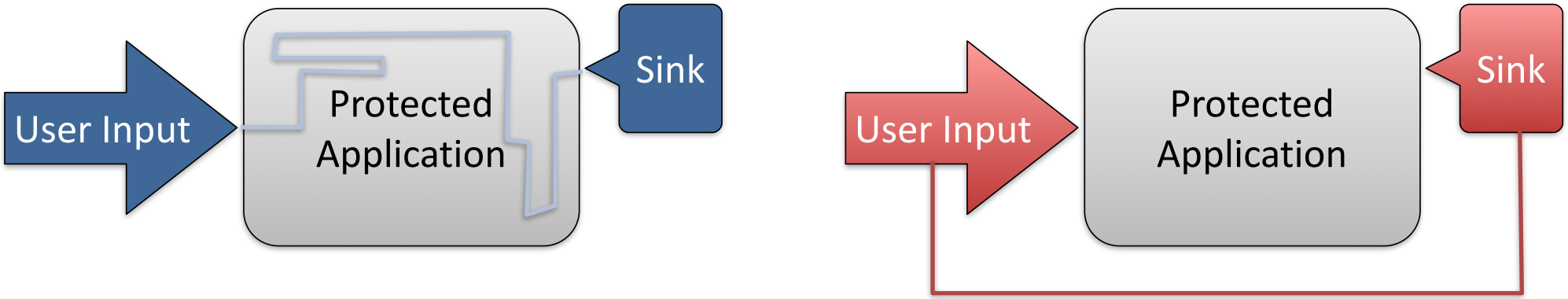
Feasibility (2)

- A typical application has 20 queries, and a few inputs.
- Queries don't typically grow very large (at most a few kilobytes), but inputs typically do.
- Specially when they upload their files
- Still in the optimum case, a polynomial of power 4 is not very fast.

Positive Taint Inference

- All discussed so far regarded negative taint inference, i.e inferring bad tainted input in the output
- Positive taint inference finds good parts of the output, inferring the rest as bad
- Remember, as long as nothing critical is bad, we're good
- Not as impossible as positive taint tracking

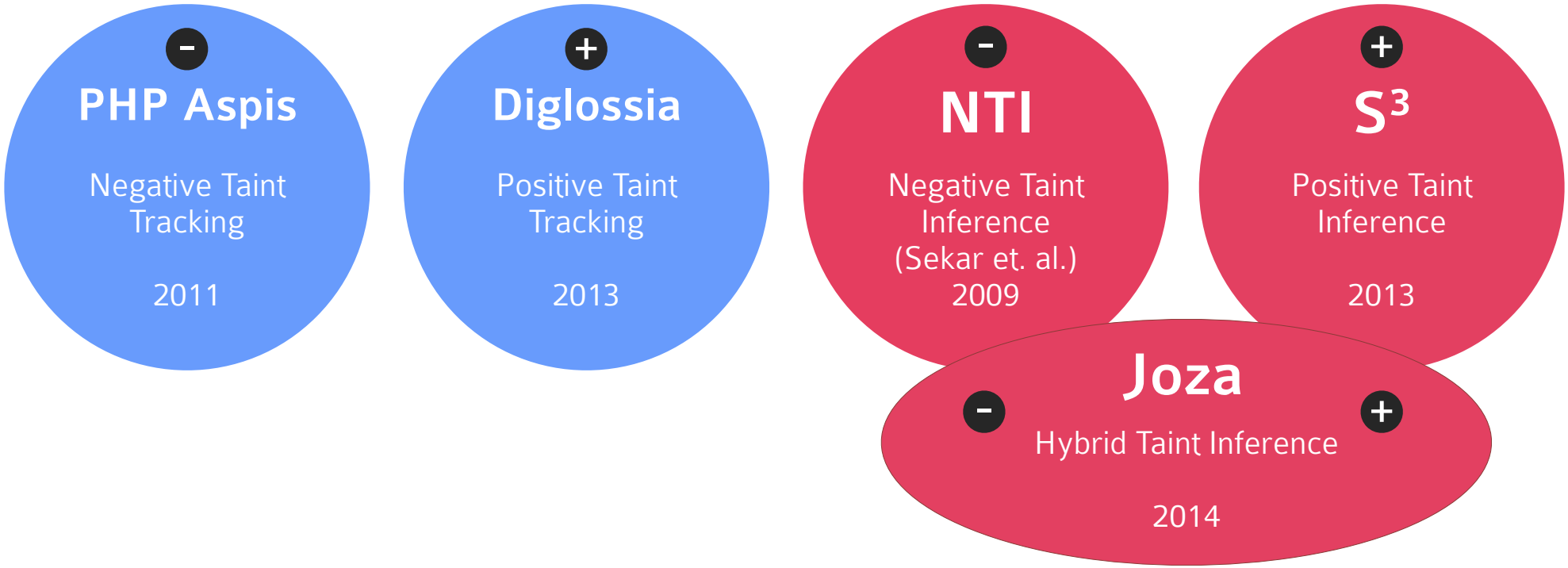
Taint-Tracking vs Taint-Inference



“Detection is, or ought to be, an exact science, and should be treated in the same cold and unemotional manner.”

Existing Techniques

We will briefly study one sample from each category:



Existing Techniques

PHP-Aspis

- Started as a taint-tracking paper
  - Turned into a PhD thesis (Imperial College folks)
- They tried to model every single function, by re-writing PHP interpreter
- There's a lot of details on how it (should) works and how they modeled everything
- But it's not actually used anywhere (last update 2011)
  - Can you guess why?

https://github.com/jpapayan/aspis

Existing Techniques

Diglossia

- Started as a positive taint-tracking paper on ACM CCS 2013
- Keeps track of user inputs, and converts application strings mixed with user-input, on a character by character basis (mapping them to Korean)
- Rewrites PHP interpreter
- At the sink, critical tokens should be Korean.
- The paper overcomplicates things to make the reader feel it's doing magic, but basically it's positive taint tracking.
- Only works on very simple operations.

Existing Techniques

NTI (by Sekar)

- Very clever method, uses negative taint inference
- Compares query at the sink with all user inputs, looking for possible approximate matches
- Uses a threshold to catch similarities
- Works pretty well to protect against trivial attacks
- Encoded input is doomed, so is transformed one
- Uses mod_security for wrapping
- Hasn't been used widely (why?)

Existing Techniques

S^3 (DNA Shotgun Sequencing)

- Uses positive taint inference
- Uses string fragments inside an application to build a query at sink
- If sensitive parts are not built by application code, they are built with user input!

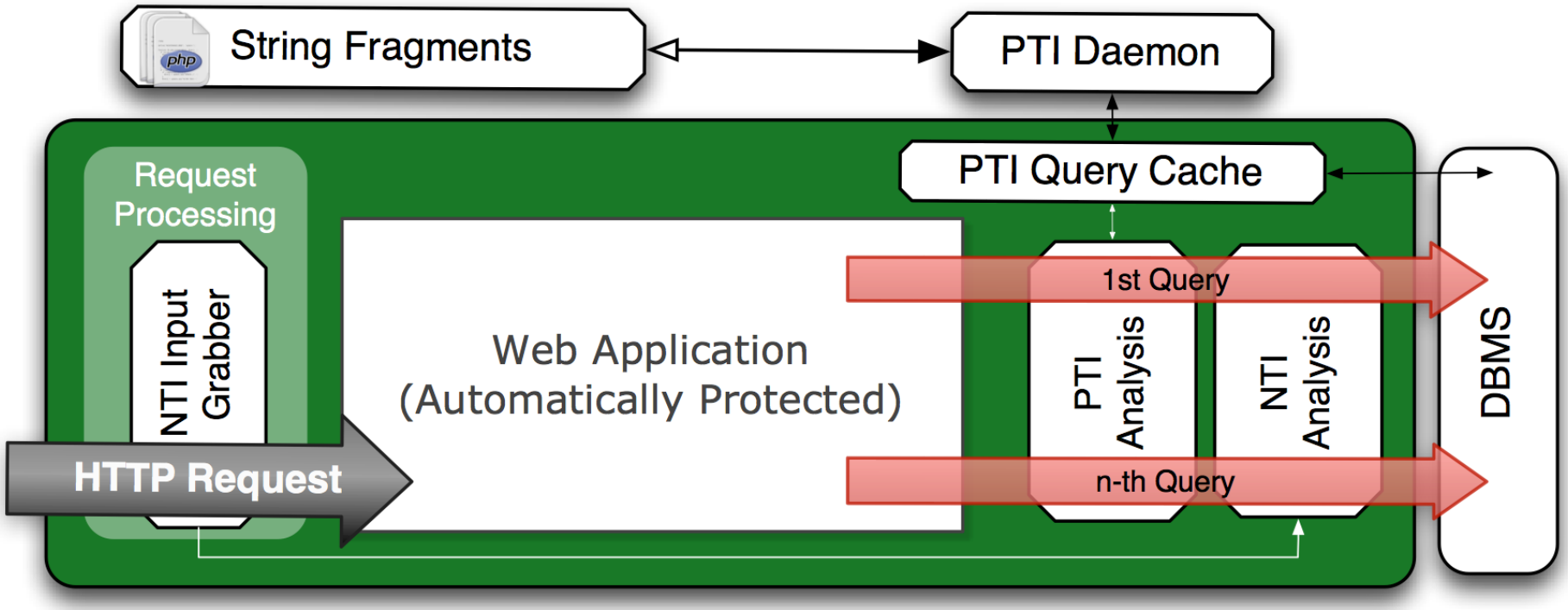
- Doesn't rely on user-input
- Doesn't rewrite PHP interpreter, instead uses a lib (or binary) and minor code modifications (one include + sink wrappings)
- Only breaks if major query parts are built dynamically (almost never)

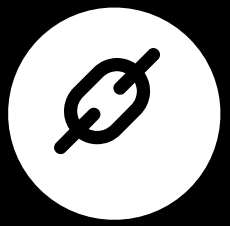
Existing Techniques

Joza

- Mixes NTI and PTI synergistically
- Very hard to break (0 false positive/negative on studies)
- Easy maintenance
- Faster NTI due to PTI
- Taintless can help break it, but just helps.
- Immune to second-order attacks

Joza Overview





“It has long been an axiom of mine that the little things are infinitely the most important.”

Taintless

Taintless Modes of Operation

Extract

Extracts plausible strings from an application as sources of positive taint

These can be used in the construct phase to build payloads that fully match positive taint sources. Not all the strings are extracted as many of them are typically used in HTML or other sources. Multiple levels of filtering and optimization is performed on the extracted strings to enable faster and more accurate processing.

Analyze

Analyzes an application, providing very useful details

Analyzes all string operations in the application code, marking hard-to-model operations as more likely to break. Breaks down application segments, suggest weak points for manual code review and the likelihood of vulnerability in the app. Detects sinks.

Construct

Constructs an attack payload using positive taint

Useful for automated scripts. Based on rigorous modified NP-complete algorithms. Even if a payload is not fully synthesized with positive taint, as much of it as possible will be covered. Requires a source of extracted fragments.

Taintless

Static Analysis

- PHP is a very irregular language
  - Impossible to amalgamate
  - Impossible to statically analyze
  - Slow to dynamically analyze
- Taintless statically analyzes a PHP application, finding possible points of failure when protecting with taint
- Analysis is based on a data file which defines how hard string operations are to model, both for tracking and inference

Sample Analysis Result

```
cust-64:PHP abiusx$ ./taintless --analyse -p ~/www/abiusx.old/ -o report.txt
Analyzing... This might take several minutes...
...............................................................................
...............................................................................
...............................................................................
...............................................................................
...............................................................................
........................
Generating statistics, please wait...
Total taint-vulnerability score of scanned path                           27698
Number of files analyzed                                                    426
Total number of sinks found                                                   3
Number of possible taint tampering functions                               2832
Taint-tracking bypass score                                               11981
Taint-inference bypass score                                              15687
PHP code elements analyzed                                              531,025

**Based on this analysis, this application is likely to break.
-------------------------------------------------------------------------------
```

Taintless

Sample Analysis Result (2)

```
Most interesting files:
# File                                                    %  Score Sinks Funcs
1 wp-admin/includes/class-wp-ms-sites-list-table.php   3.0%  819    0    16
2 wp-admin/includes/upgrade.php                        2.6%  720    0    37
3 wp-includes/ms-functions.php                         2.3%  650    0    48
4 wp-signup.php                                        2.2%  616    0     9
5 wp-admin/edit.php                                    2.1%  585    0     7
6 wp-admin/includes/update.php                         1.9%  528    0     6
7 wp-admin/upload.php                                  1.7%  468    0     6
8 wp-admin/plugin-editor.php                           1.7%  468    0     8
9 wp-admin/includes/class-wp-filesystem-ssh2.php       1.6%  450    0    23
-------------------------------------------------------------------------------
Most important sinks:
# File                                                    %  Score Sinks Funcs
1 wp-includes/wp-db.php                                0.1%   38    2    24
2 wp-includes/SimplePie/Cache/MySQL.php                0.1%   19    1     1
```

Taintless

Extraction

- Parses every single file in an application, extracting strings
- Placeholder strings (e.g printf format string, PHP inner-concat) are broken down into multiple strings
- The final list of strings is filtered for those with SQL (or any other attack) tokens, and the rest are discarded
- Strings with binary (terminating) characters are discarded
- The list is sorted and duplicates removed

Sample Extraction Result

```
cust-64:PHP abiusx$ ./taintless --extract -p ~/www/abiusx.old/ -o abiusx.fragments.txt
Extracting text fragments, this might take up to several minutes...
..........................................................................
..........................................................................
..........................................................................
..........................................................................
..........................................................................
.........................
Filtering fragments to find those of interest... This might take very long...
[core] Total fragments: 39837, interesting (SQL-Like) fragments: 8908; dumping to abiu
sx.fragments.txt...

Done.
```

Taintless

Construction

- Solves modified maximum coverage problem (NP-Complete) to build a string with available fragments in an application
- Whitespaces and comments are extended and/or shrieked for better matching results
- Possible forms of a token are all searched for (e.g union all, union)
- SQL payload is not parsed as it is not a full query, the user is in charge of determining if all critical tokens are matched
- SQLMap tamper script included

Sample Construction Result

```
cust-64:PHP abiusx$ ./taintless --construct -i abiusx.fragments.txt -s "1' union
 all select 1,2,database() -- "
Filtering duplicate fragments... Succesfully filtered 1186 duplicates!
Matching fragments... done.
Resolving 3 confusions... done.
Resolving 2 confusions... done.
Found all useful bits and pieces, gluing... done.
Coverage: .✓.......✓✓✓.✓✓✓✓✓..✓.✓.........✓✓.✓✓
Result  : 1' union All Select 1,2,database() --
Pieces  :  1       222 333333  4 5        66 77
```

Taintless

Sample Construction Result (2)

```
cust-64:PHP abiusx$ ./taintless --construct -i abiusx.fragments.txt -s "1 and 1=
(select mid(password,1,1) from users where username='admin') "
Filtering duplicate fragments... Succesfully filtered 1186 duplicates!
Matching fragments... done.
Resolving 6 confusions... done.
Resolving 5 confusions... done.
Resolving 3 confusions... done.
Resolving 2 confusions... done.
Resolving 1 confusions... done.
Found all useful bits and pieces, gluing... done.
Coverage: ..///..////////....//////////././/.////......./////.....//////.//////
Result  : 1 AND 1=(Select mid(PASSWORD,1,1) From users where username='admin'
```

Q&A

Questions?

Abbas Naderi (aka AbiusX)
Mandana Bagheri
Shahin Ramezany

Challenge Wall of Fame

Siavash Mahmoudian
Mykola Ilin
Shivam Dixit
Mathias Bynens
Abouzar Parvan
Ahmad Moghimi
Mohammad Teimori Pabandi

Solve me?

abiusx.com

1 A Special Thanks To

University of Virginia, ZDResearch, OWASP, Etebaran Informatics and all others that made development of this tool possible.

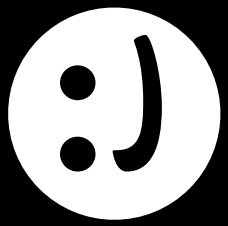
2 Follow Us Twitter:

Twitter:
AbiusX
ZDResearch
OWASP Iran
Shahin Ramezany

We will be hosting a CTF with taint-protected challenges soon, cash prizes included!

3 Test Taintless Yourself

WP-SQLI-LAB and WP-SQL-SINK tools provide a great test-bench for Wordpress SQL injection. Simplified implementations of Taint Tracking, NTI and PTI are available, and detailed implementations can be obtained by emailing respective authors.



“There is nothing new under the sun. It has all been done before.”