

# Winning the Online Banking War

---

**Sean Park**  
**Senior Malware Scientist**  
**TrendMicro**

# About Myself

---

- Worked for one of big 4 banks in Australia for 6 years as malware security consultant.
- Developed banking malware detection system
- Served at Sophos, Symantec, FireEye and Kaspersky
- Currently with TrendMicro

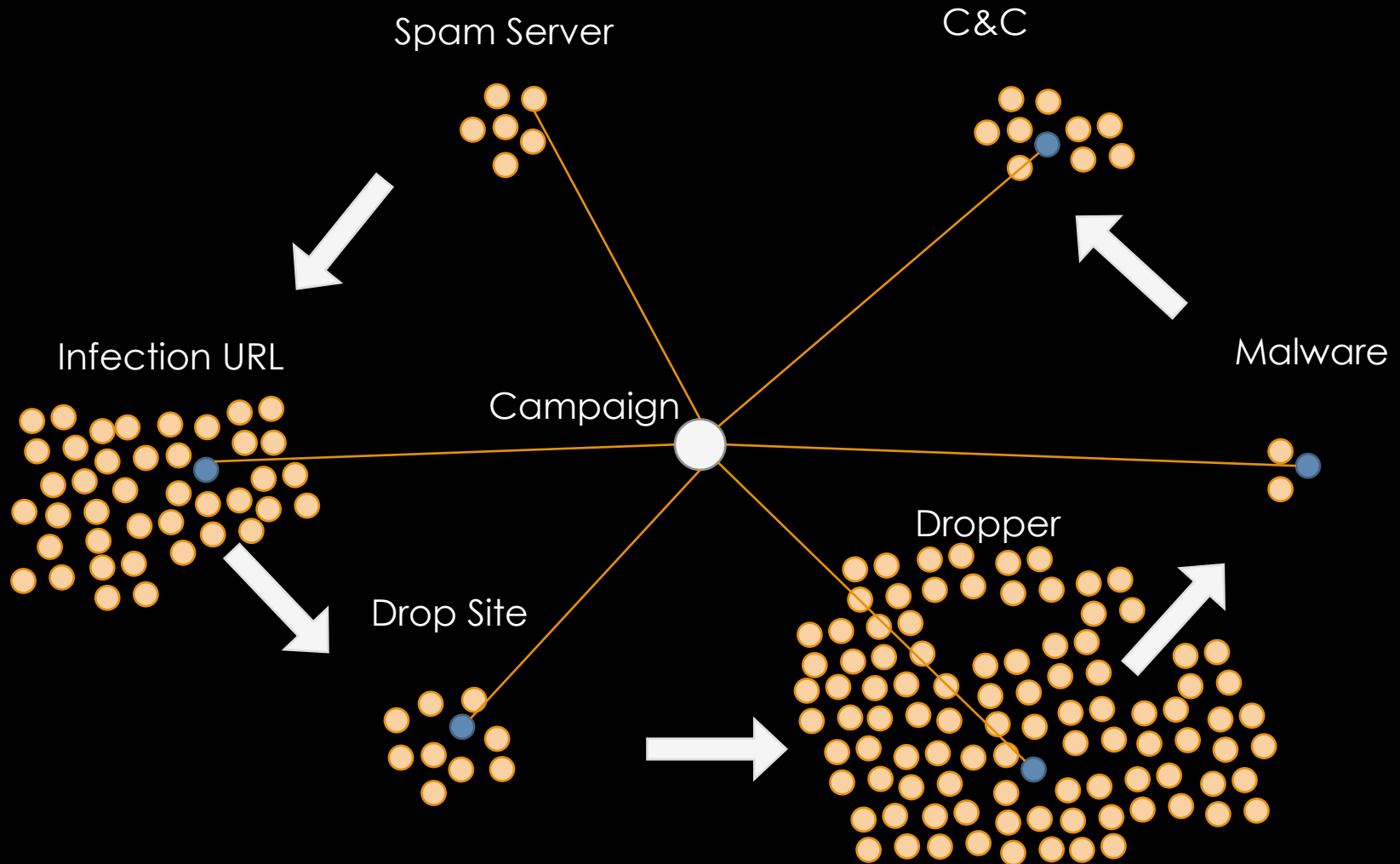
# Goals

---

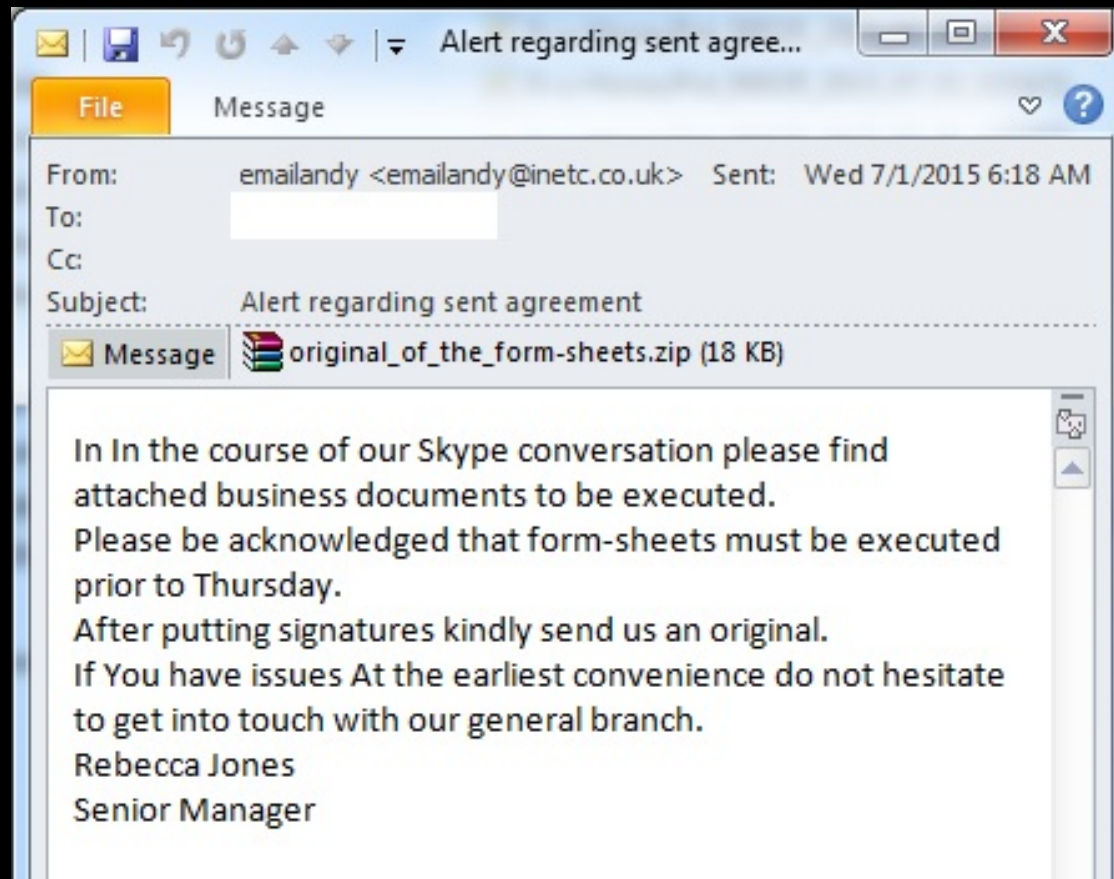
- Identify the crux of the online banking war
- Set the strategic defense framework
- PoC design & implementation: MIPS

# Banking Malware Campaigns

---



# Customer Infection



# Sign In – Phone Banking Code

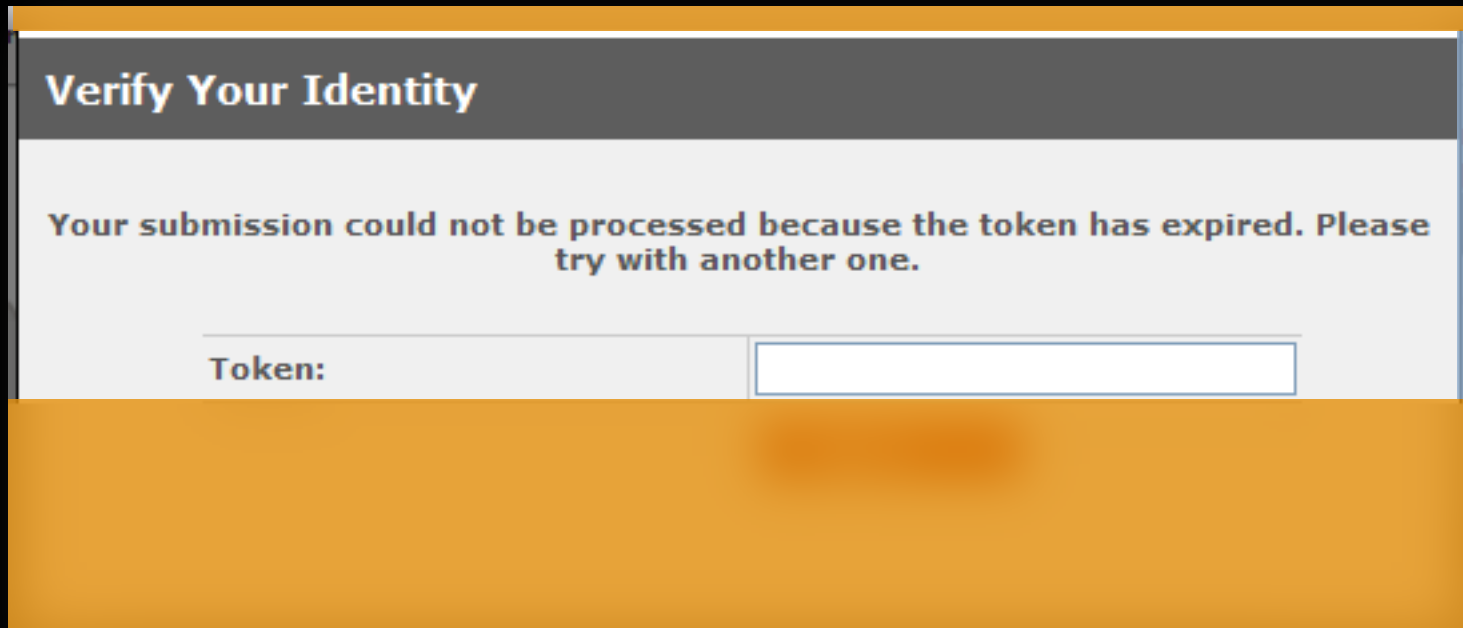
At the moment, is the process of gathering unique data on your system to create a unique digital signature (UDS). In the future the system will identify your computer by UDS. Please enter the following information:

|             |                                      |   |
|-------------|--------------------------------------|---|
| <b>code</b> | <b>Your telephone banking access</b> | <input type="text"/>  |
|             | <b>Your date of birth</b>            | <input type="text" value="dd"/> / <input type="text" value="mm"/> / <input type="text" value="yyyy"/> |
|             |                                      | <input type="button" value="Clear"/> <input type="button" value="Submit"/>                            |

# Sign In – Token

---

- Got a token for your corporate account? Do you still feel safe?



**Verify Your Identity**

Your submission could not be processed because the token has expired. Please try with another one.

Token:

The image shows a web form titled "Verify Your Identity" with a grey header. Below the header, a message states: "Your submission could not be processed because the token has expired. Please try with another one." At the bottom of the form, there is a label "Token:" followed by an empty text input field. The entire form is set against a light blue background.

# Sign In – Token

---

- Now you are locked out while they buy enough time to transfer money

## Verify Your Identity

**XXXXXXXXXXXXXXXXXXXX is temporarily unavailable. This is due to Scheduled Maintenance to enhance your online service. The XXXXXXXXXXXXXXXX apologises for any inconvenience that this may cause. System will be available within 4 hours.**



# Sign In - MITM

---

- There is no such a thing as 'Please Wait' in the online banking page.



The image shows a screenshot of an online banking sign-in page. It features two input fields: 'Customer No.' with a partially obscured value ending in '17', and 'Password' with a masked input. Below these fields is a large orange rectangular area, likely a placeholder for a logo or image. To the right of this area, a button labeled 'Please Wait ...' is highlighted with a red rectangular border. The button contains a small circular icon with a loading symbol (three dots) and the text 'Please Wait ...'.

# Sign In - MITM

- What's happening while you are waiting...

```
ofsrgnqqapfpvlxz.org /news/
olutions.es /fotos/dbs_res.exe
online. /
ofsrgnqqapfpvlxz.org /news/
pass.com /script.js?i=1
www.google.com /webhp
pass.com /script.js?r=0.9535408292260581&1=75&2=WJ7&url=https%3A%2F%2F
pass.com /script.js?r=0.20854243438889675&aid=784
pass.com /script.js?r=0.30924708325910066&aid=784
pass.com /script.js?r=0.4451089154071417&aid=784
pass.com /script.js?r=0.9629884590830888&aid=784
pass.com /script.js?r=0.5519197805007762&aid=784
pass.com /script.js?r=0.425544130092404&aid=784
pass.com /script.js?r=0.2673889011694235&aid=784
pass.com /script.js?r=0.012008610301909084&aid=784
```

```
pass.com /script.js?r=0.012008610301909084&aid=784
pass.com /script.js?r=0.012008610301909084&aid=784
pass.com /script.js?r=0.012008610301909084&aid=784
pass.com /script.js?r=0.012008610301909084&aid=784
```

# Transaction Injection - SMS

---

**You have 1 incomplete transaction between your accounts.**

**Amount: \$3,500.01**

**Bank Operator: ~~Service Advisor~~**

**If you would like to cancel this payment or consider that it happened by mistake, please type in the secure code below.**

**Your Secure Code was sent to #### ##9 977 via Voice Call**

Secure Code

APPROVE TRANSACTION

CANCEL TRANSACTION

# Transaction Manipulation

---

- Even when there is no visual sign of infection, it can happen silently.
- C&C communication during Tx pages

```
online. [redacted] translist.asp?acctref=0
[redacted] webanalytics.com /public/wp_/global
online. [redacted] getdetails.asp?FunctionID=7
[redacted] webanalytics.com /public/wp_/global
online. [redacted] getdetails.asp?FunctionID=11
[redacted] webanalytics.com /public/wp_/json
[redacted] webanalytics.com /public/wp_/global
[redacted] webanalytics.com /public/wp_/details
[redacted] webanalytics.com /public/wp/bt?bid=12&dt=%5B%7B%22n%22%3A%22Damian%2
online. [redacted] confirm.asp
[redacted] webanalytics.com /public/wp_/global
[redacted] webanalytics.com /public/wp_/confirmcorp
[redacted] webanalytics.com /public/wp/hm
```

# Transaction Manipulation

- What is the malware receiving? → Inject and Mule

```
HTTP/1.1 200 OK
Server: nginx/0.7.67
Date: Wed, 18 Apr 201
Content-Type: text/html
Connection: keep-alive
X-Powered-By: PHP/5.2.17
P3P: CP="NOI ADM DEV PSAi COM NAV OUR OTRo STP IND DEM"
Expires: Thu, 19 Nov 198
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Vary: Accept-Encoding,User-Agent
Content-Length: 120
```

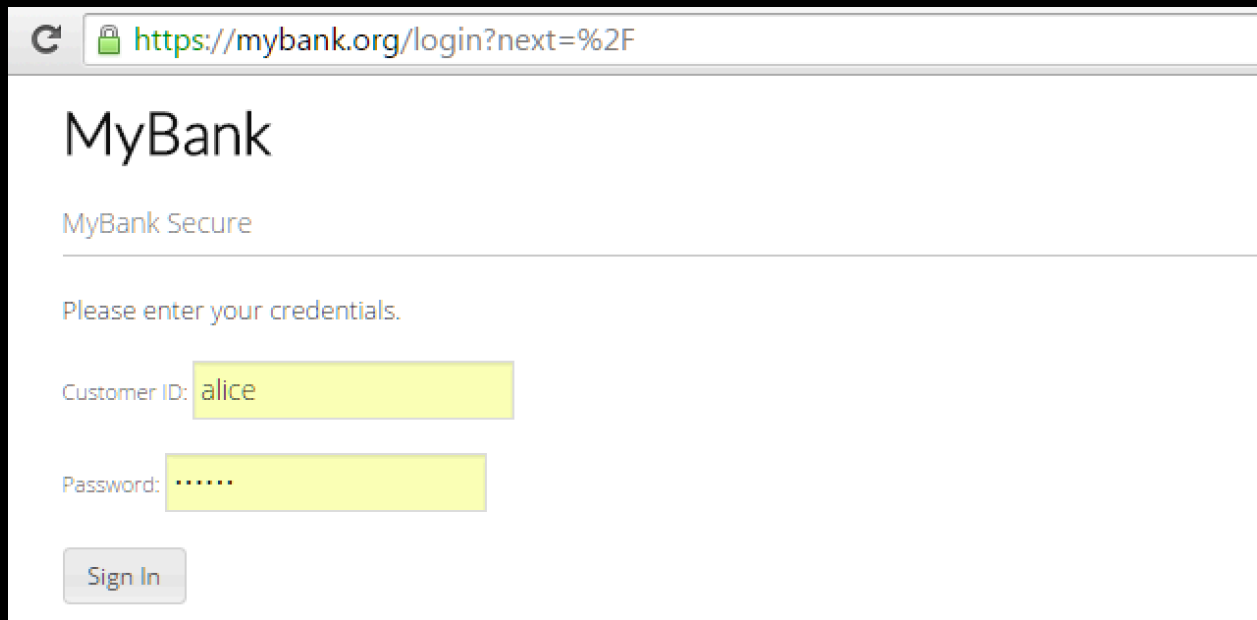
```
drcvd(["b": "9", "a": "22", "n": "N e", "s": "4500.00", "ob": ")", "oa": "53", "on": "Dam"]])
```

Mule's Account  
Information

Transfer Amount

# Phase I DOM Injection

# Attack: DOM Injection



MyBank

MyBank Secure

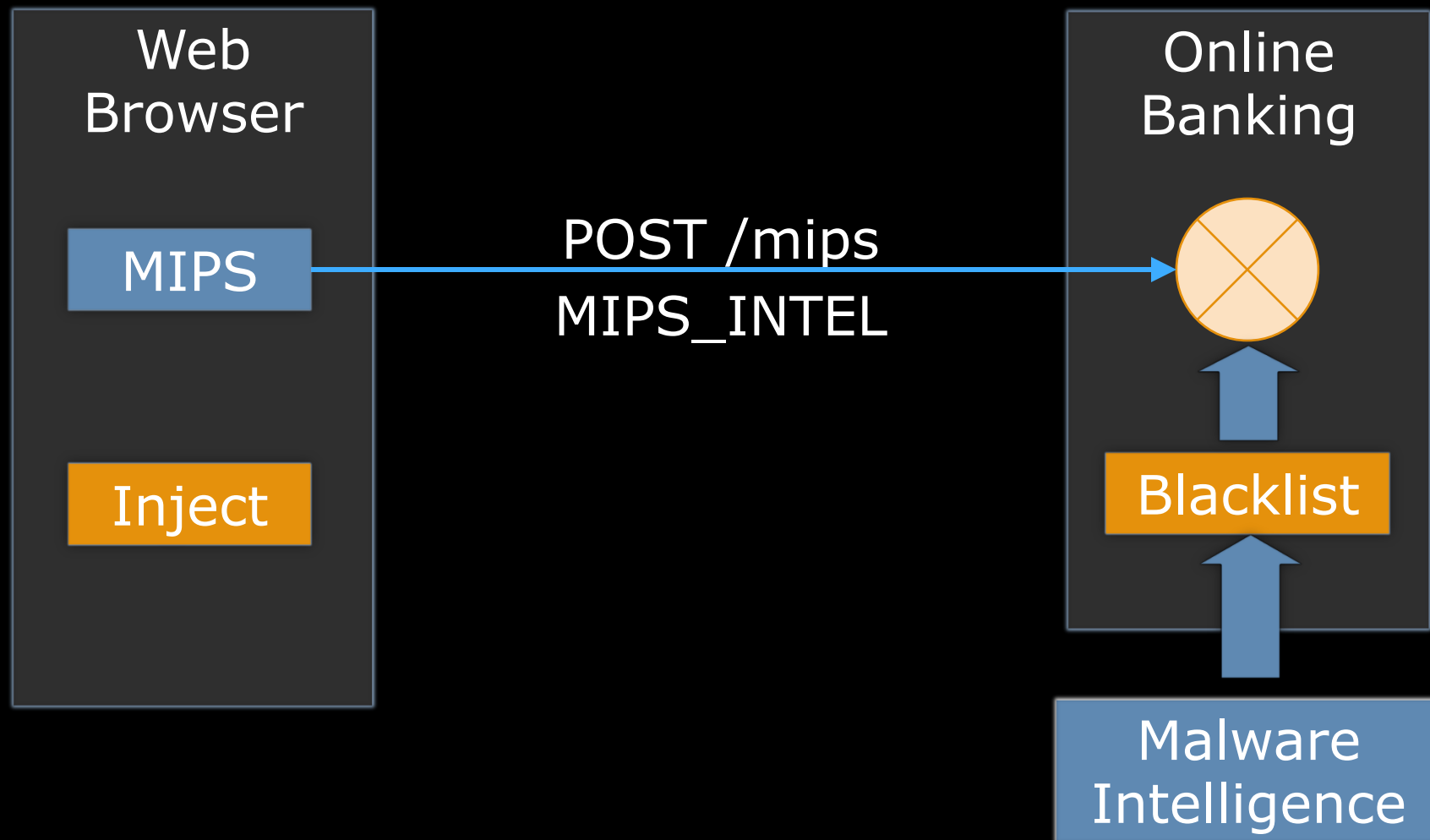
Please enter your credentials.

Customer ID:

Password:

```
$("#submit").on("click", function(){  
    var id = $("#signin-id").val();  
    var pw = $("#signin-password").val();  
    console.log(">> DOM Inject: "+id+": "+pw);  
});
```

# Defense: DOM Scan





# Phase II DOM Stealth

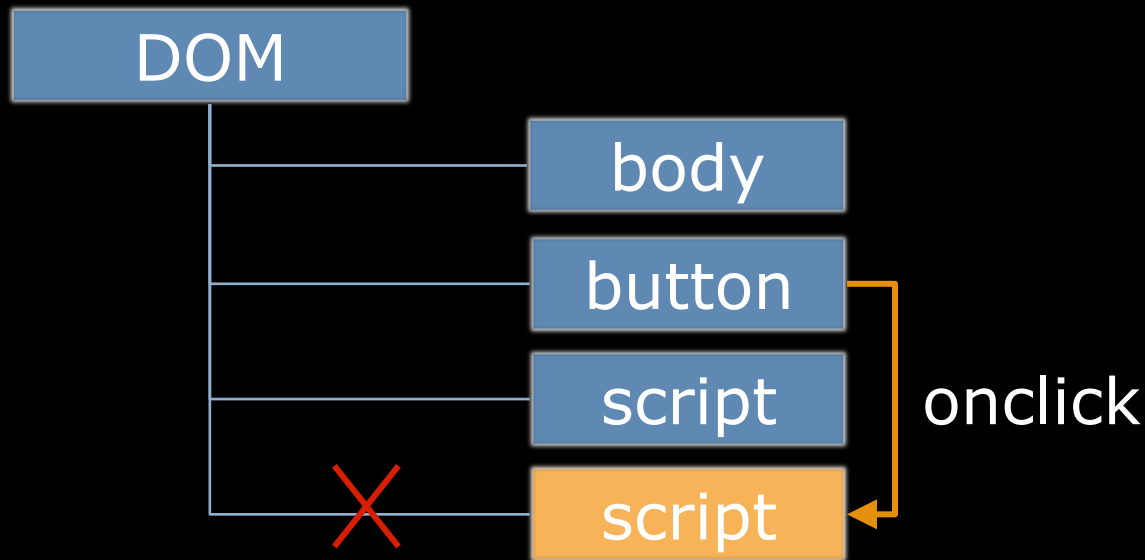
# Attack: How It Works

---

- Malware inject removes itself, but it still remains in the memory
- Exploit memory leak patterns
  - Dangling references
  - Circular references
  - Closures

# Attack: Dangling References

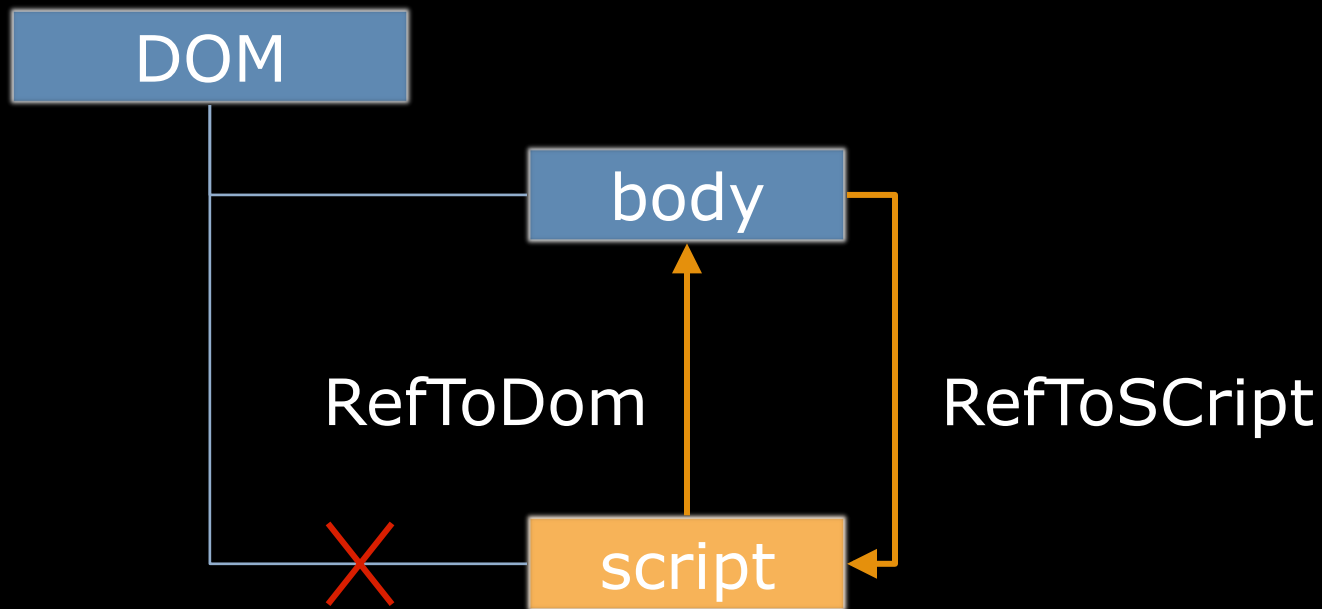
---



```
var me = document.currentScript;  
me.parentNode.removeChild(me);
```

# Attack: Circular References

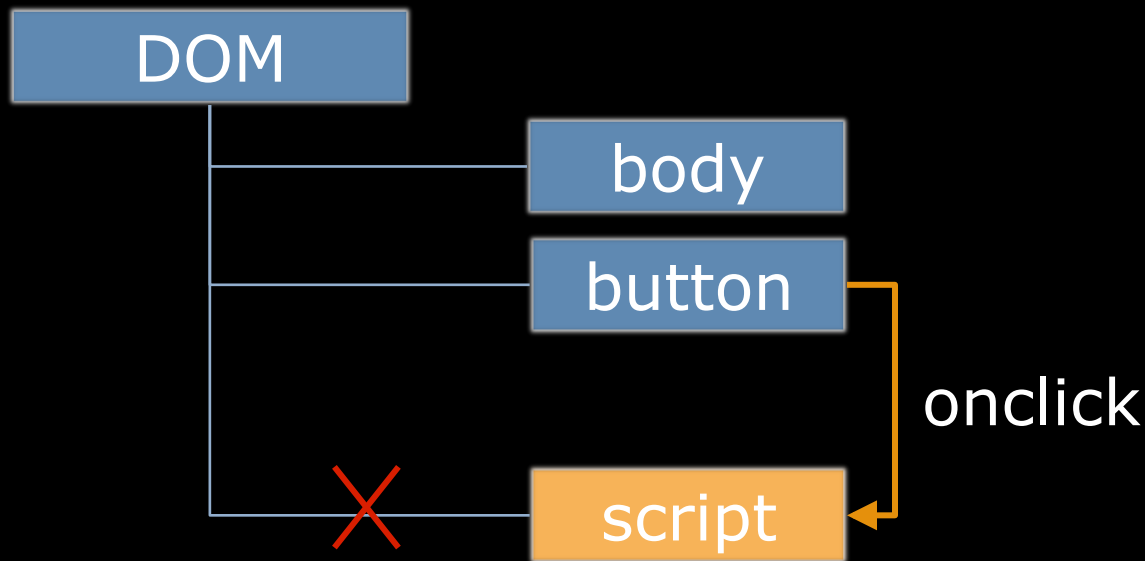
---



```
var refToDom = document.body;  
document.body["refToScript"] = refToDom;
```

# Attack: Closures

---



```
function AttachEvent(element) {  
    element.attachEvent("onclick", MyClickHandler);  
    function MyClickHandler() {  
        /* This closure references element*/  
    }  
}
```

# Defense: DOM Event Scan

---

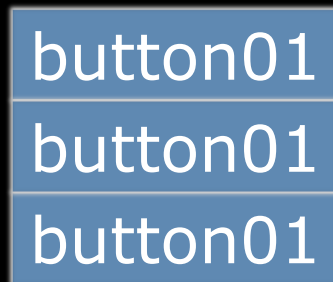
- Identify entry points (unload, click, timer)
- Enumerate event handlers
  - element.onclick = handler*  
Scan: `element.onclick`
  - element.addEventListener*  
Scan: `getEventListeners(element, "click")`
  - \$(element).on("click", handler)*  
Scan: `$.data(element, "events" )`
  - \$(element).observe("click", handler)*  
Scan: `element.getStorage().get('prototype_event_registry').get('click')`

# Defense: Artefact Analysis

---

For `$('#submit'), 'click'`,  
Event handler's **namespace** property is missing

Normal

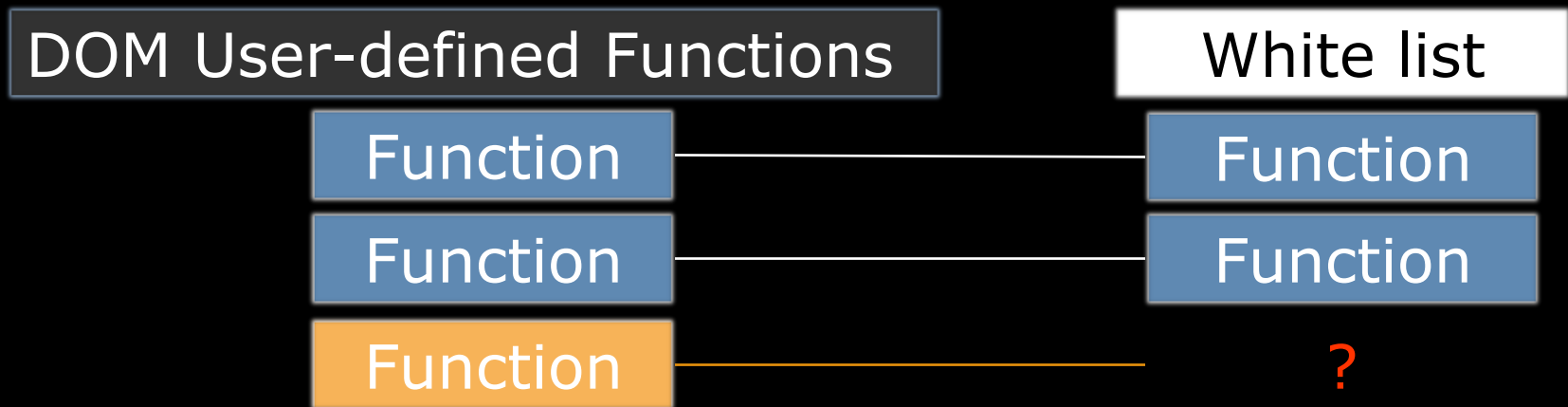


DOM Stealth



# Defense: Function Integrity Check

---

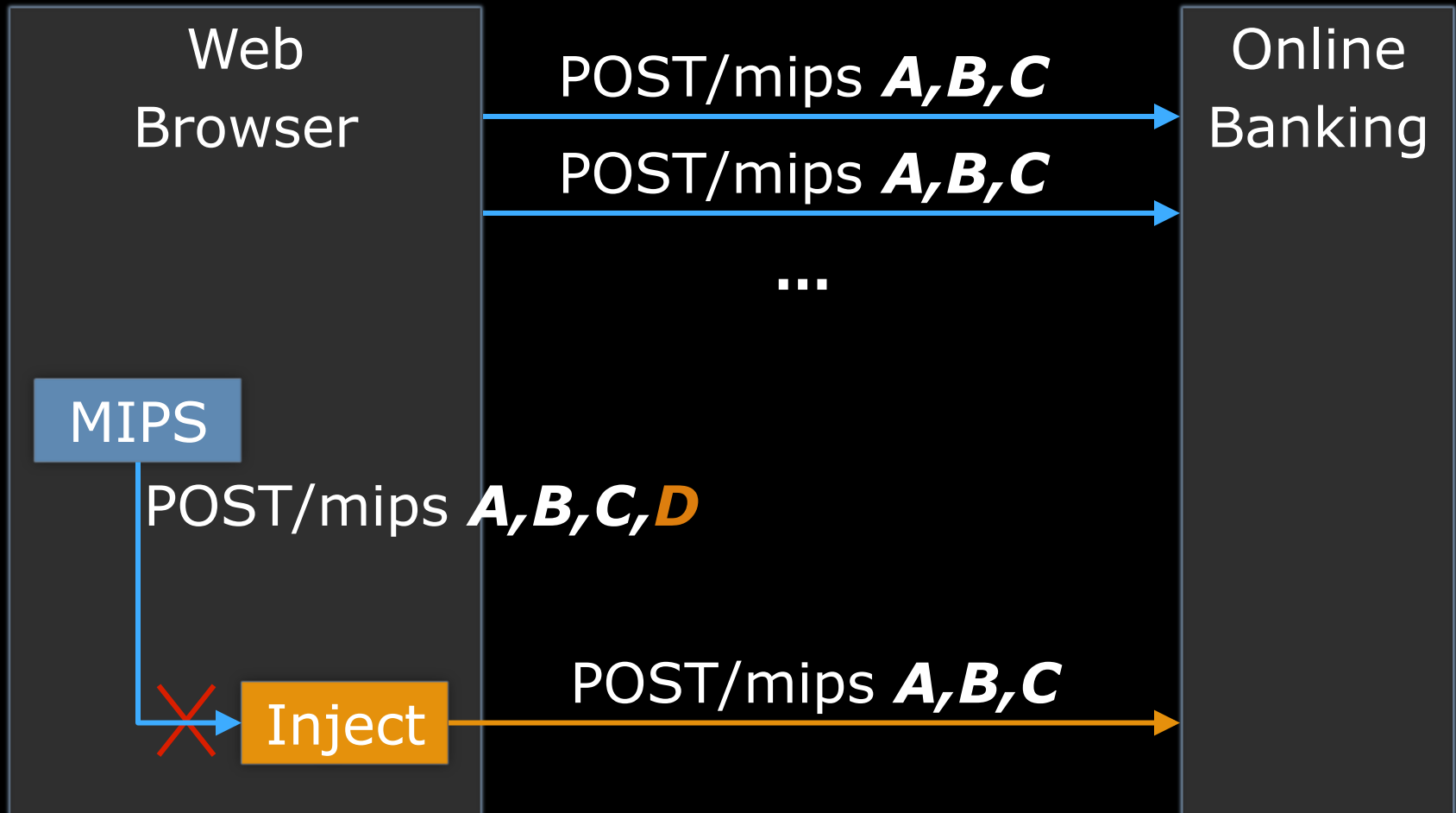


- Enumerate user-defined functions
  - `Object.keys(window).filter( !/[native code]/ )`
- Compare functions discovered in DOM against whitelist
- Implementation challenges
  - Check on server side or client side?
  - Reducing data size

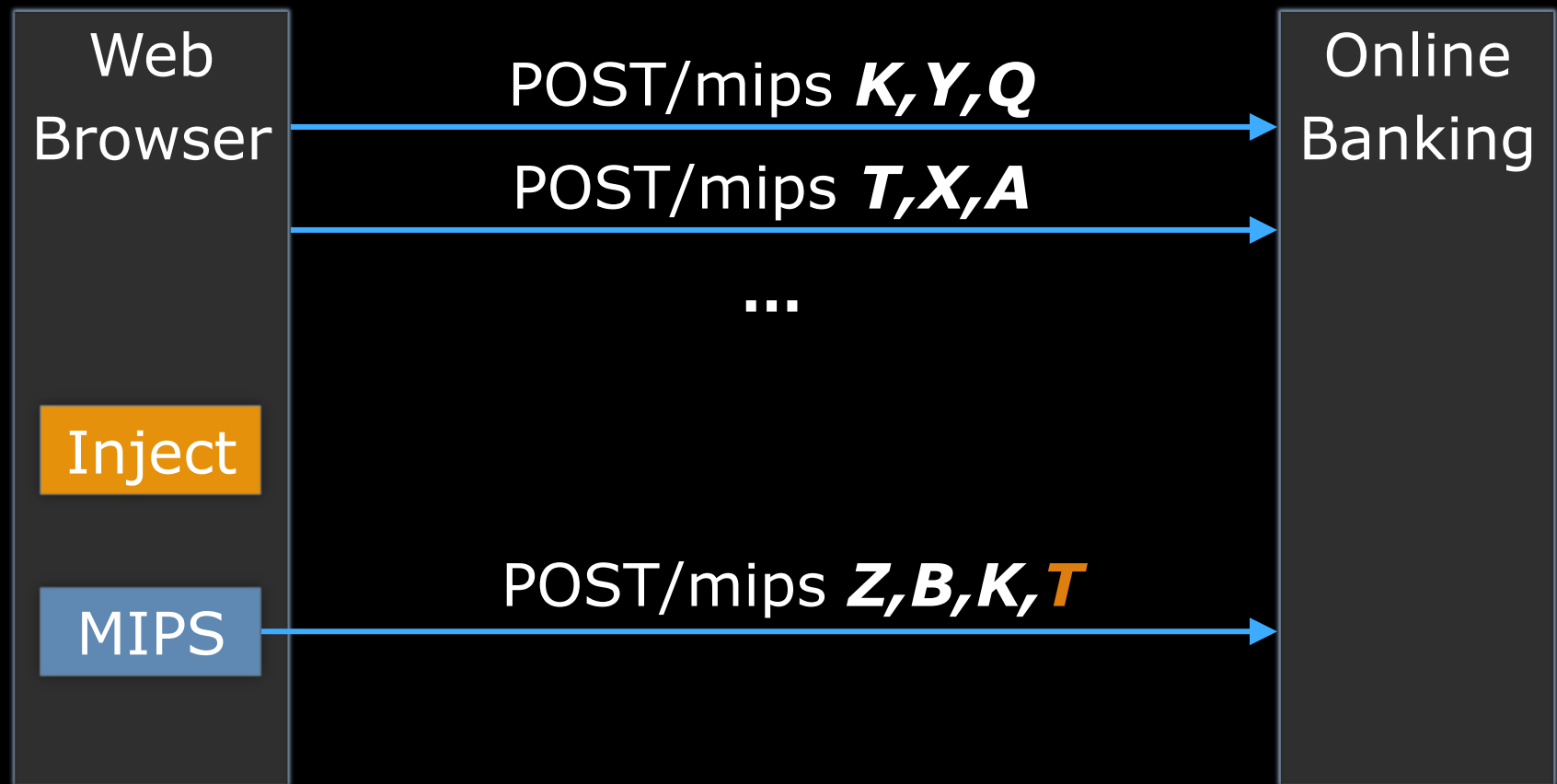


# Phase III MIPS Infiltration

# Attack: Replay



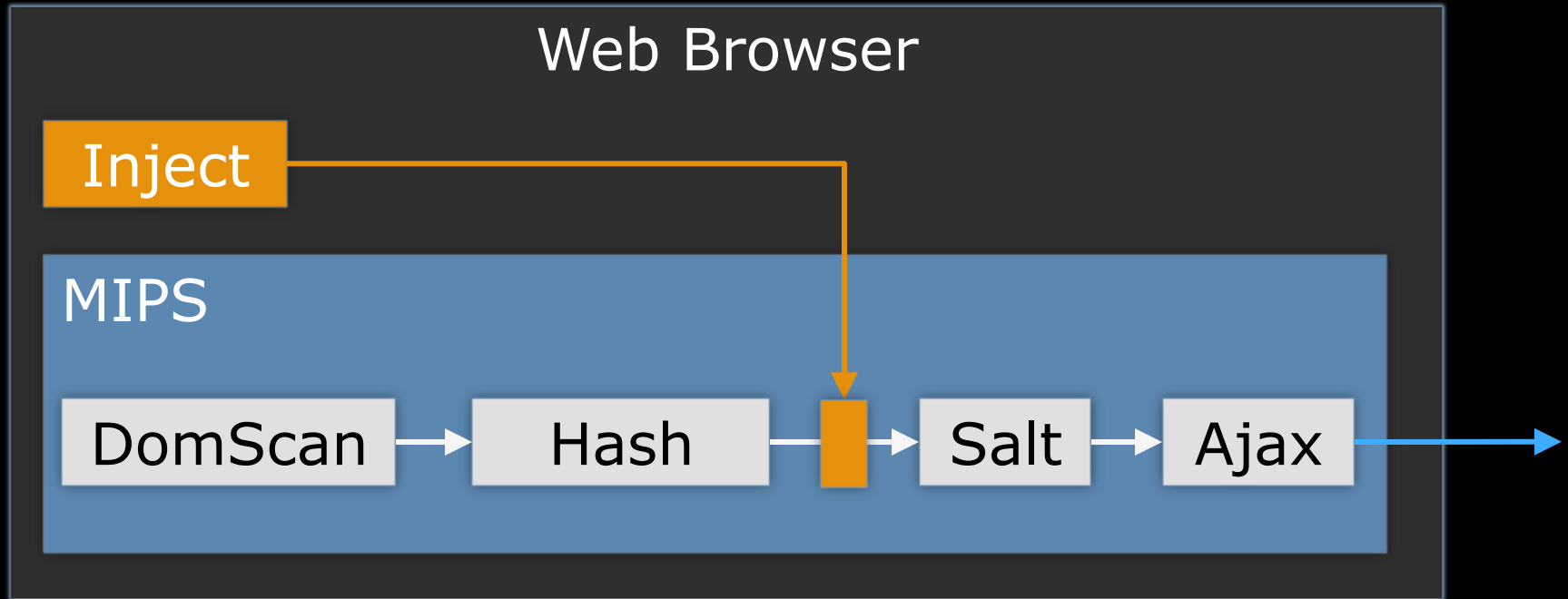
# Defense: Salting



- Original MIPS intel gets transformed differently each time using the random variable

# Attack: Forging MIPS Intel

---



- Hook Salt() and modify hashes OR
- Block MIPS & call MIPS functions as necessary

# Attack: Model 1

---

- Insert or replace with bypass code within MIPS code
- Tamper with intermediate MIPS data

```
var scripts = DomScan(mips_code);  
scripts = Modify(scripts);  
var hashes = Hash(scripts);  
var salted_hashes = Salt(hashes);  
var check = CheckIntegrity();  
check = Modify(check)  
Ajax(mips, salted_hashes, check);
```

# Attack: Model 2

---

- Reverse engineer MIPS
- Deactivate MIPS and Reconstruct MIPS code

```
var scripts = DomScan(mips_code);  
var hashes = Hash(scripts);  
var salted_hashes = salt(hashes);  
var check = CheckIntegrity();  
Ajax(mips, salted_hashes, check);
```

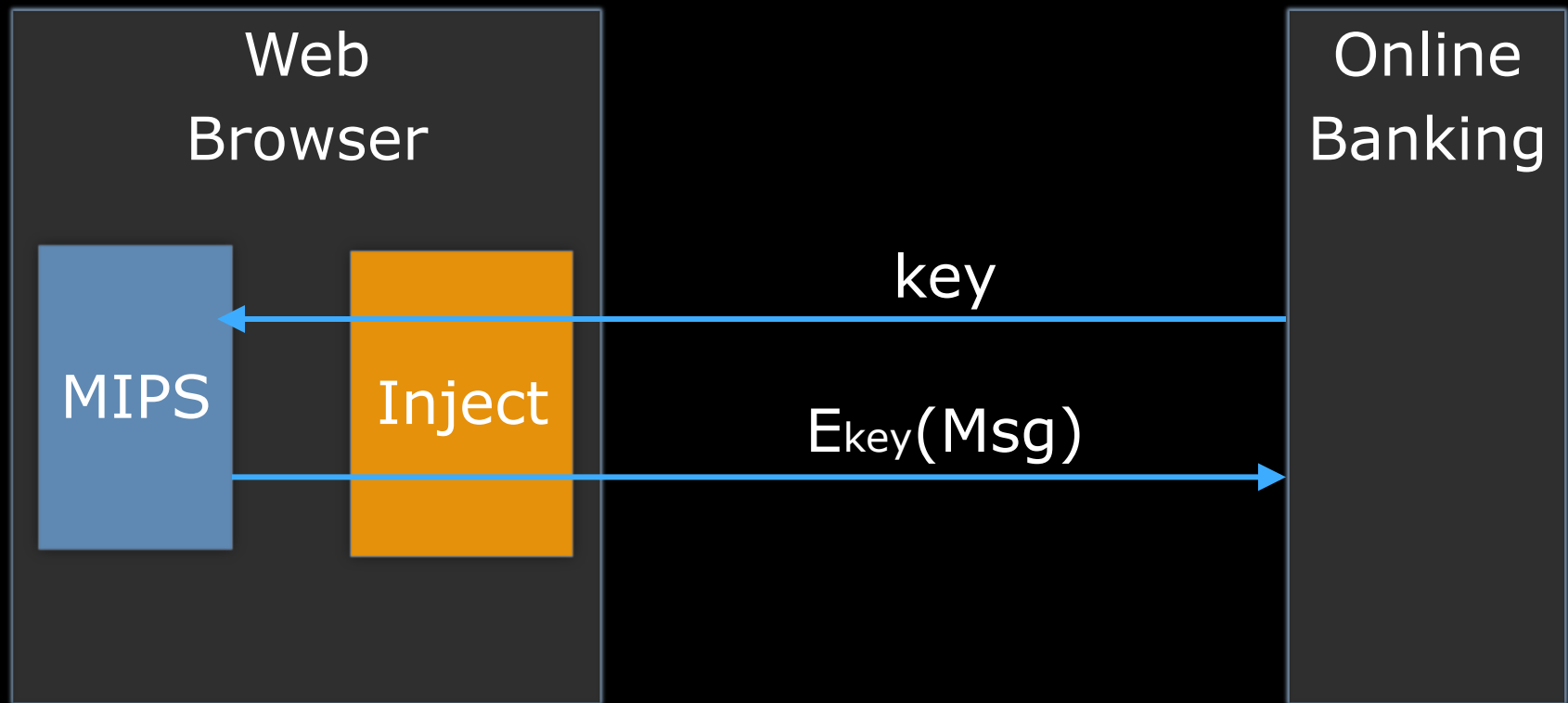
```
var hashes = clean_hash_set;  
var salted_hashes = salt(hashes);  
Ajax(mips, salted_hashes, clean_integrity_check);
```

# Bad Defense: Obfuscation

---

- What you are up against
  - Dynamic analysis
    - Use static analysis tools (Google closure compiler, spider monkey, custom tools, etc)
    - Understand program structure by setting breakpoints and evaluating expressions
    - Bypass dead code
    - Monitor network traffic
    - Targeted reverse engineering by searching keywords (i.e. 'script', '/mips')
  - Activity monitoring
- Blind obfuscation is not resistant to targeted code inspection/modification

# Bad Defense: Encryption



- Malware has all info to simulate outcome
  - key, encryption algorithm



# Bad Defense: More

---

- ***Blind application*** of traditional metamorphic/polymorphic techniques without understanding the attack vector will fail
  - Dead code insertion
  - Polymorphic variable/function names
  - Control flow manipulation
  - Function restructuring
  - Opaque predicate
  - etc

# Defense: Code Integrity Check

---

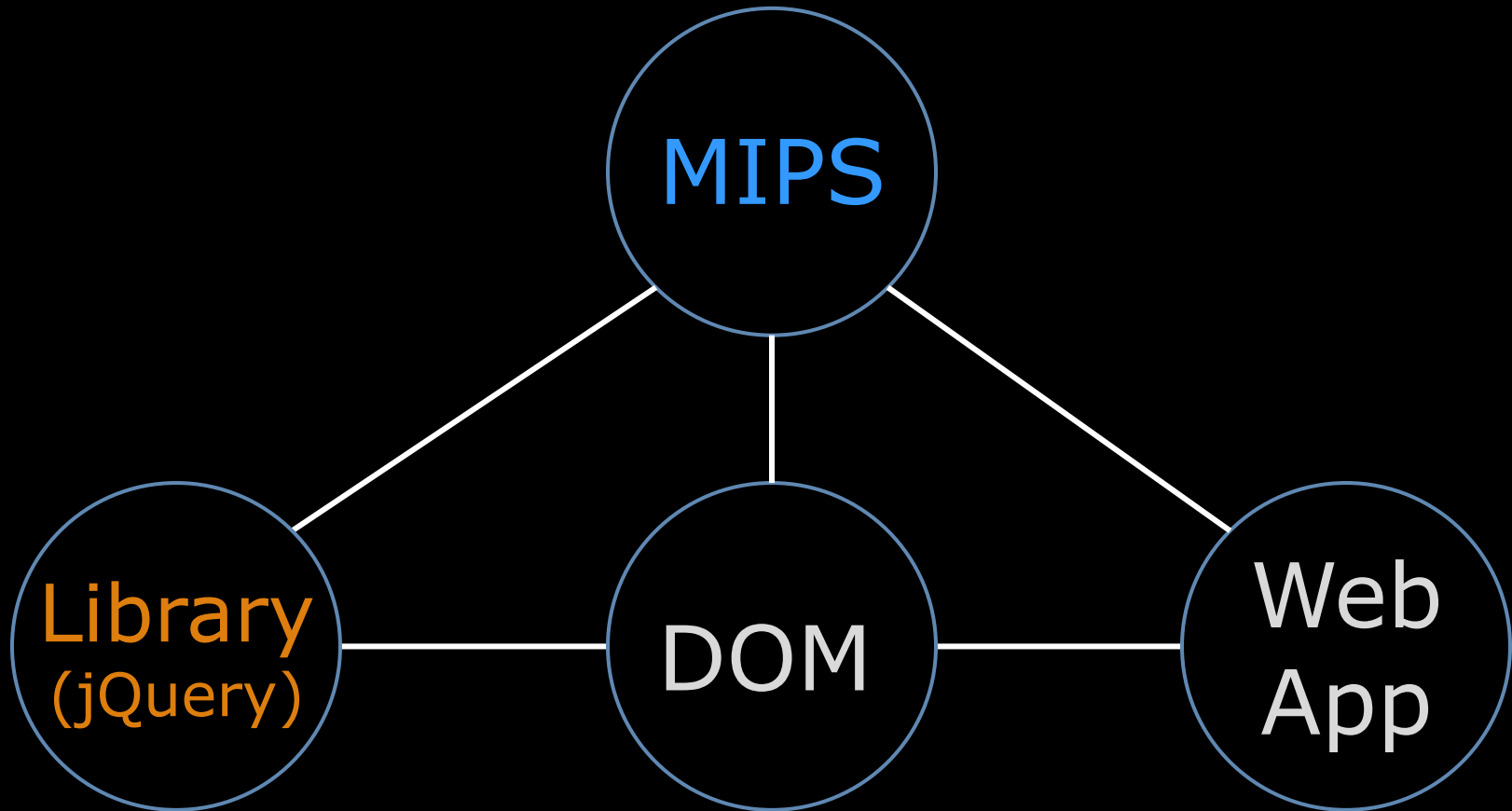
- Call stack context

```
var Check = function(na, nb) {  
  var SecureCheck = function(na, nb) {  
    var callee = na ^ crc32(arguments.callee);  
    var caller = nb ^ crc32(arguments.callee.caller);  
    return callee ^ caller ^ DomCheck();  
  };  
  return SecureCheck(na, nb);  
};
```

```
var na = 32053221, nb = 4321053;  
result = Check(na, nb);
```

# Defense: Code Integrity Check

---



# Defense: Randomisation

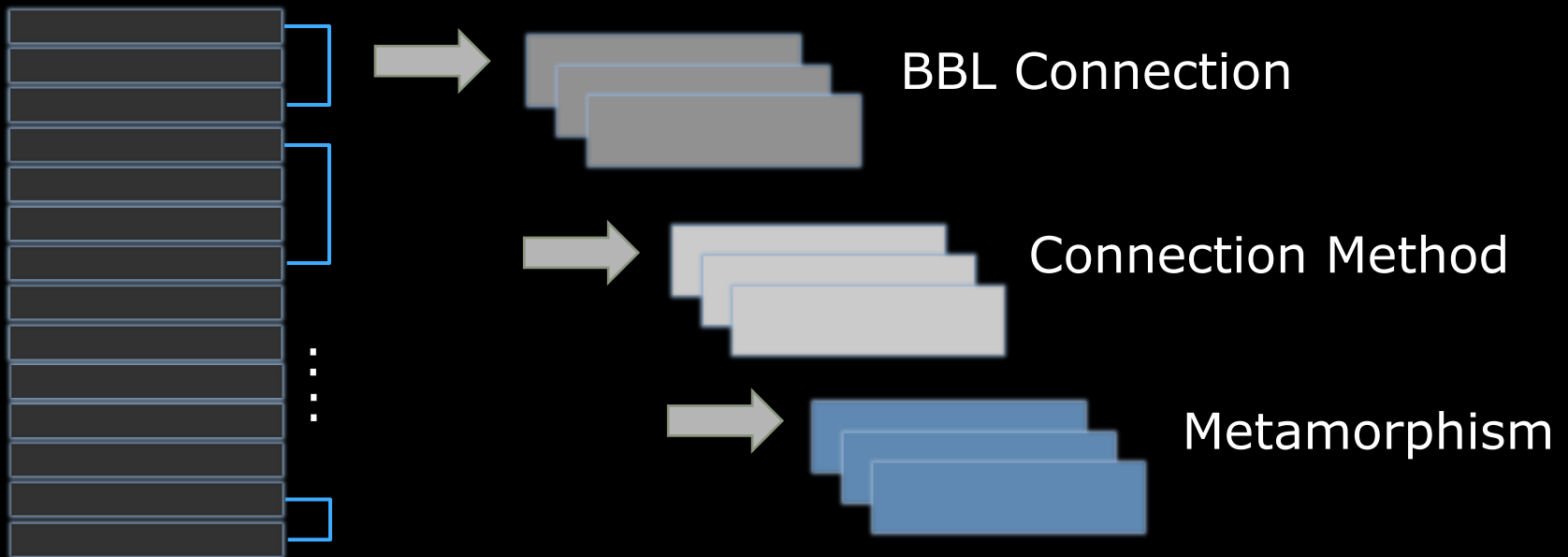
---

- Problem with integrity check
  - Malware Regexes, modifies and reconstruct MIPS
  - Malware simulates MIPS with bypass code
- Strategy
  - Polymorphism
  - Maintain a set of **algorithmically heterogeneous** MIPS code
  - Fragmented random MIPS scripts with different names

# Defense: Control Flow Randomisation

---

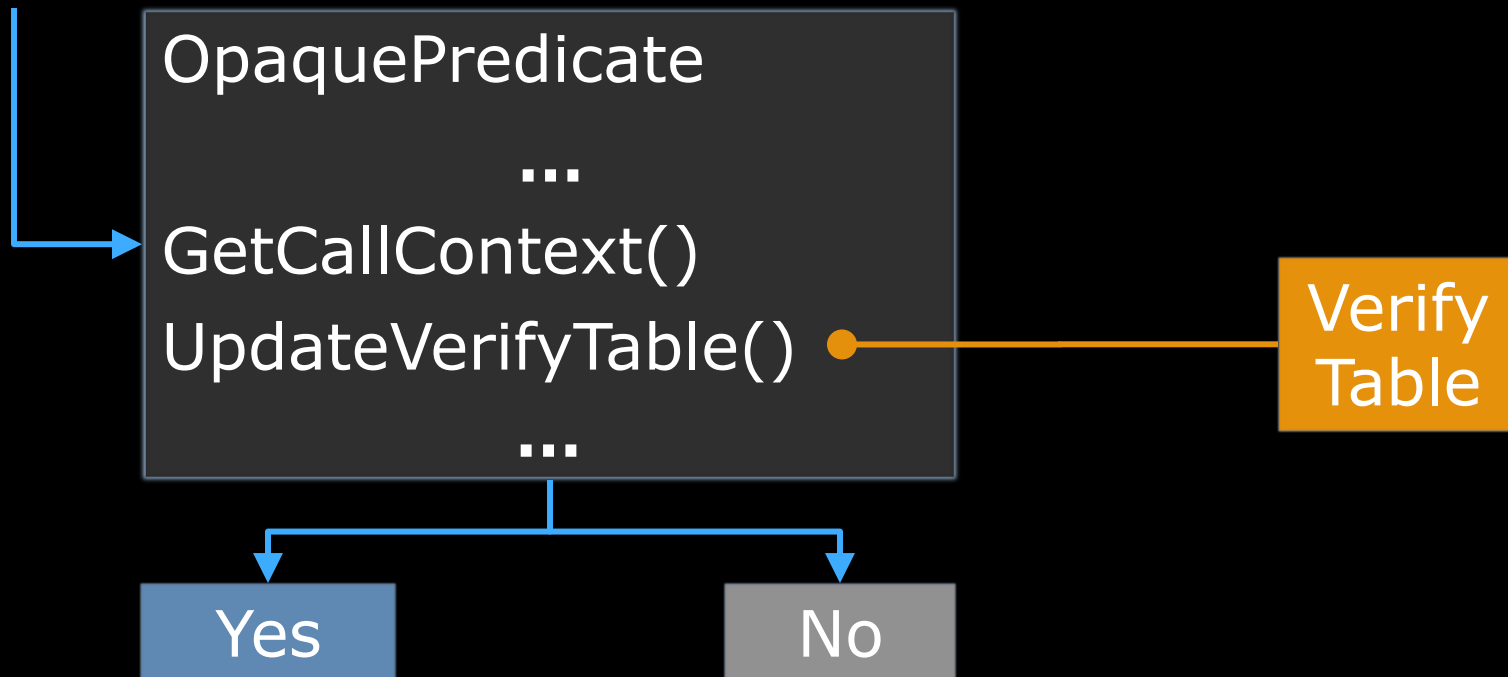
MIPS BBLs



- Chain of Randomisations starting with basic blocks (BBLs) of MIPS code

# Defense: Opaque Predicates

---



- Retrieve call context in deeply buried OP
- Insert part of main logic within OP

# Phase IV Rootkit

# Attack: How It Works

---

```
var original_func = document.getElementsByTagName;
document.getElementsByTagName = function () {
  r = original_func.apply(document, arguments);
  for(var i=0; i<r.length; i++) {
    var inject_signature = 'string_in_my_inject';
    if(r[i].text.search(inject_signature) != -1) {
      r[i].remove();
      console.log('Injeject Rootkitted!');
      break;
    }
  }
  return r;
};
```



# Attack: How It Works

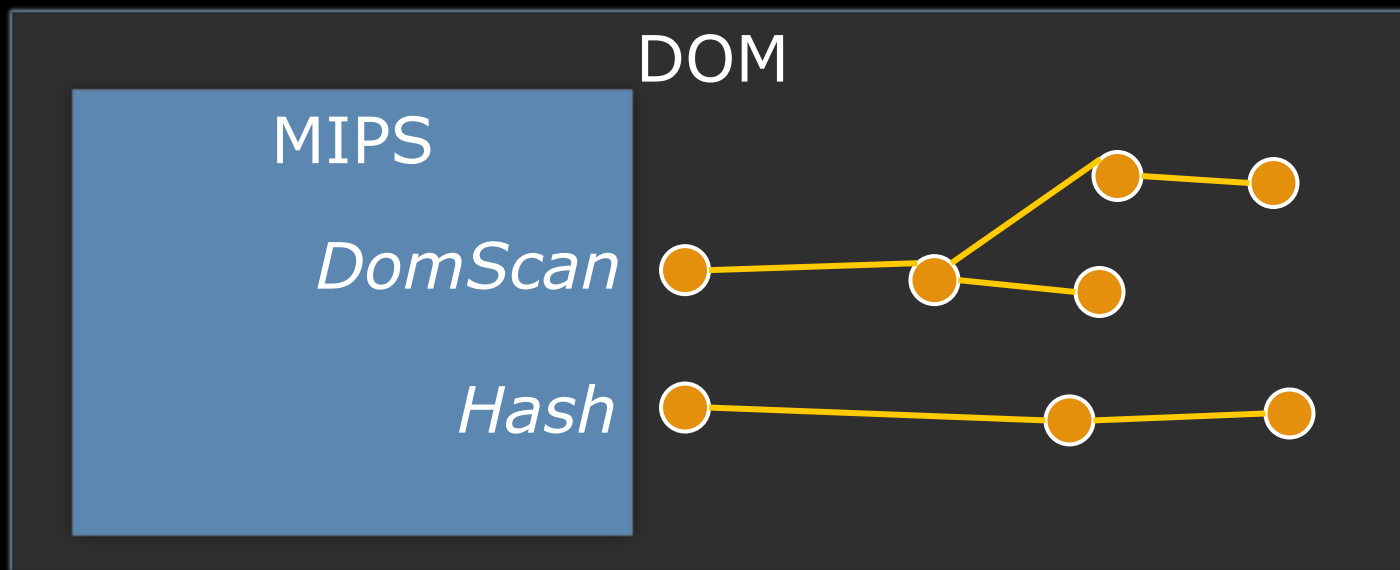
---

- Hook critical MIPS functions
  - DomScan(), Hash(), Salt()
- Hook DOM/jQuery
  - *document.getElementsByTagName(selector)*
    - Modify the returned HTMLCollection
  - *jQuery.find(selector, doc, ret)*
    - Modify the returned array
- Hook system information
  - *Object.keys()*
  - *Function.prototype.toString()*

# Defense: DOM Integrity Check

---

- Collect signatures of chain of functions used by MIPS in multiple different ways.
- Online banking server detects any change



# Defense: Detecting Rootkits

---

- Deliberately trigger exception → Call stack

```
var hooked = Function.prototype.toString;
Function.prototype.toString = function() {
    hooked.apply(this, arguments);
} // DOM Rootkit
```

```
var TriggerException = function(){
    try {
        Function.prototype.toString.call('hooktest')
    }
    catch(err) {
        console.log(err.stack);
    }
}
TriggerException();
```

# Defense: Detecting Rootkits

---

- Is the red line present in a clean session?

```
TypeError: Function.prototype.toString is not  
generic
```

```
  at String.toString (native)
```

```
  at String.Function.toString(/login?next=%2F:173:7)
```

```
  at TriggerException (/login?next=%2F:177:29)
```

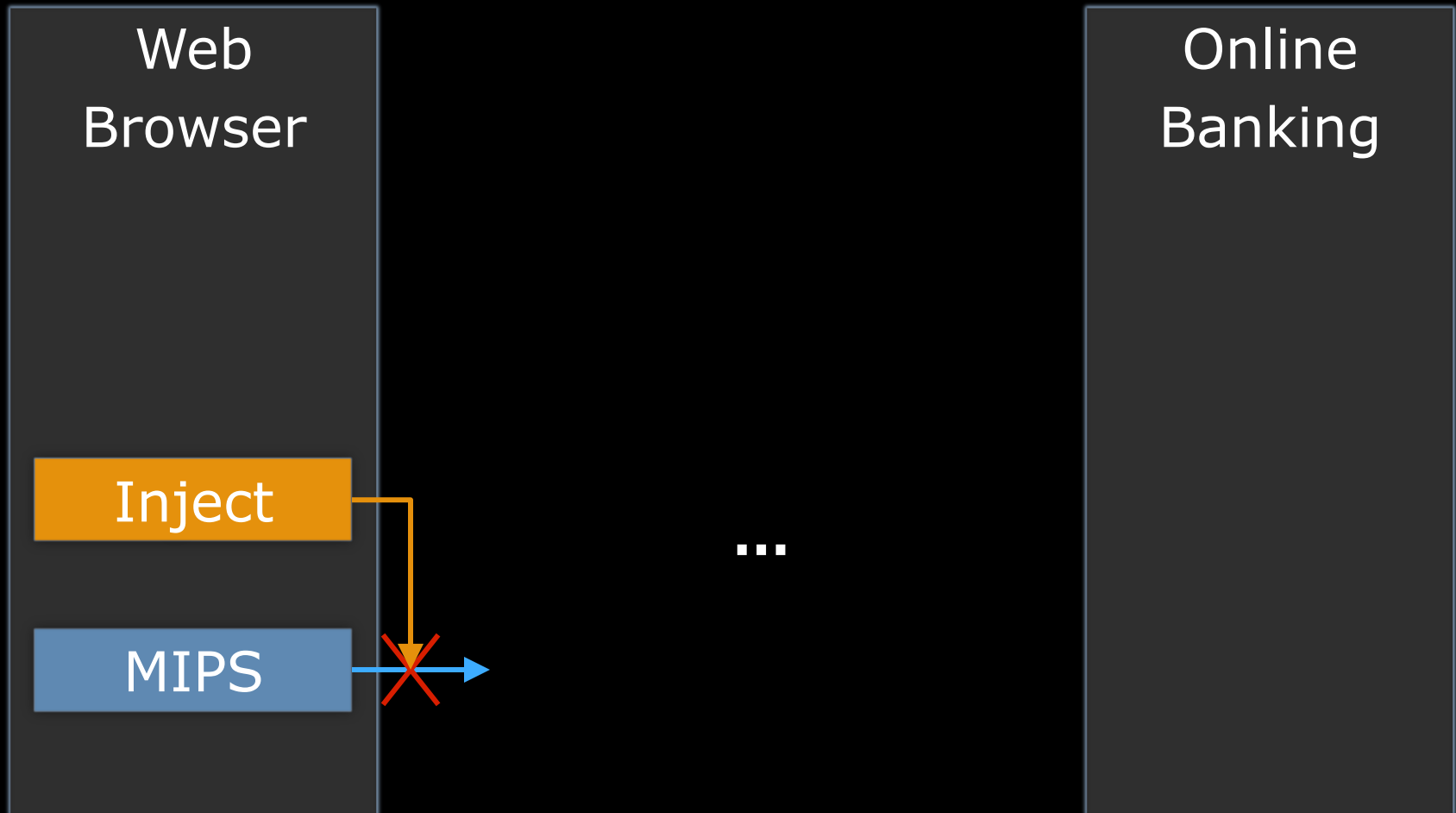
```
  at https://mybank.org/login?next=%2F:183:1
```

# Phase V

## Fraud Analytics

# Attack: Blocking MIPS

---



# Defense: MISSING\_MIPS Event

---

- MISSING-MIPS Event should be implemented on the online banking server side if MIPS is not the integral part of online banking logic
- Method
  - Ensure MIPS intel is not cached by the proxy in-between
  - Correlate web access log with MIPS log

# Detecting Moving Targets

---

- Detect evolving injects
  - Effective on minor inject upgrade
- Methods
  - Locality sensitive hashing (i.e. TLSH)

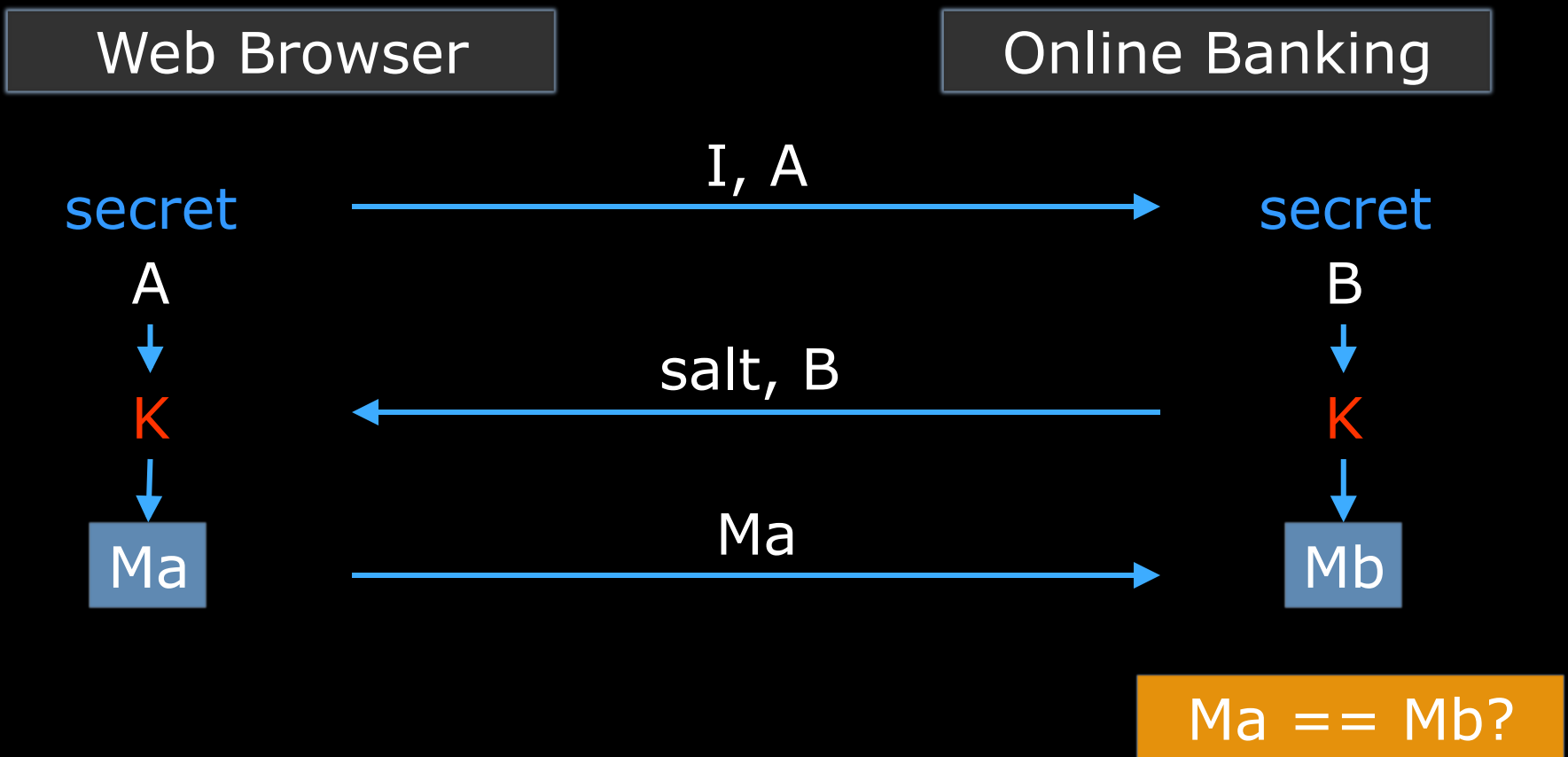


# Phase VI

## Zero Knowledge Proof

# ZKP: SRP

- Over-simplified Secure Remote Password



# ZKP: SRP

---

- No secrets on the wire any more!

[/mips/zkp\\_start?I=text\\_14&A=2ccaf4d78a5ad576907d7bbf17bba358f3...](/mips/zkp_start?I=text_14&A=2ccaf4d78a5ad576907d7bbf17bba358f3...)  
[/mips/zkp\\_verify?sessionid=13470271979590360666996200509990162...](/mips/zkp_verify?sessionid=13470271979590360666996200509990162...)

▼ Query String Parameters [view source](#) [view URL](#)

|    |   |
|----|---|
| I: | text_14   |
| A: | 7325c0ef4d3eb0778085d9a9ac801776ab06fe6d69<br>6a688ce74e203ae2a1c5bedf54e0e20749070f23062392d |
| r: | 1436183361904   |

▼ Query String Parameters [view source](#) [view URL](#)

|            |  |
|------------|--|
| sessionid: | 8175930623352600519908330954227903320    |
| M1:        | f55caf69584086906f4395edda0532724c1bf650 |

# Use Cases

---

- MITM attack
  - No shared secrets get transmitted on the wire (password, OTP code)
- Passive sniffing
  - Force attackers to place injects (so we can detect it!)
- MIPS hardening
  - DOM function integrity data
  - MIPS integrity data
  - MIPS rootkit detection data
  - MIPS intelligence format

# Conclusion

---

- Diversity of implementation is the key for survival
- Be creative and out-smart the cybercriminals!
- Perform application security check
- Never explicitly block on the spot on detection!

Live Demo

MIPS

# Black Hat Sound Bytes

---

- Majority of the attacks presented came from the observation in real online banking war
- DOM stealth, rootkit and MIPS infiltration are a natural evolution of the attack
- Online banking must respond with superior defense including at least code randomisation, MIPS integrity verification and rootkit detection

# Thank You

---

**Sean Park**  
**Senior Malware Scientist**  
**TrendMicro**