Dynamic Spyware Analysis

Egele, Kruegel, Kirda (TU Vienna), Yin, Song (CMU)

Nicolas Boichat, Eric Bisolfati

May 7, 2009

Outline

Introducing the terminology - Spyware

Spyware: Malicious code installed on victim's machine:

- **Monitors** the behaviour of users (URL visited, passwords...)
- **Leaks** the data back to the attacker (saved to disk, sent over the network)

Introducing the terminology - BHO

Browser Helper Objects (BHO):

- DLL loaded by Internet Explorer (share its address space)
- Normally provide **extensions** to IE (pop-up blocking, mouse gestures...)
- Also used by about **75% of spywares**

Spyware Analysis Approach

Common practise:

- Detection through **signature detection**
  $\longrightarrow$ Database must be kept up-to-date
  $\longrightarrow$ Difficulty to deal with obfuscation

Goal:

- Dynamic analysis
- Static analysis
- Determine **which** sensitive data is leaked and **where** it is sent

Spyware Analysis Approach

3 complementary approaches:

- **Dynamic Taint Analysis**: Sensitive data is tainted
  $\longrightarrow$ Dynamic analysis
- **Operating System Awareness**: Distinguish between actions performed by IE and by BHO
- **Browser session Recording and Replaying**: IE visits a large number of pages (test coverage)
  $\longrightarrow$ Spyware will trigger and reveal its malicious behaviour

Spyware Analysis Approach

3 complementary approaches:

- **Dynamic Taint Analysis**: Sensitive data is tainted
  $\longrightarrow$ Dynamic analysis
- **Operating System Awareness**: Distinguish between actions performed by IE and by BHO
- **Browser session Recording and Replaying**: IE visits a large number of pages (test coverage)
  $\longrightarrow$ Spyware will trigger and reveal its malicious behaviour

System Overview

Dynamic taint analysis:

- Built on top of QEMU (**assembly code analysis**)
- Sensitive data marked as **tainted**, and tracked through the browser and BHO
- If tainted information is leaked by BHO outside the address space of the browser:
  $\longrightarrow$ **malicious flow**
- If leaked by IE:
  $\longrightarrow$ **"benign" flow** (history tracking, saving in cache...)

System overview

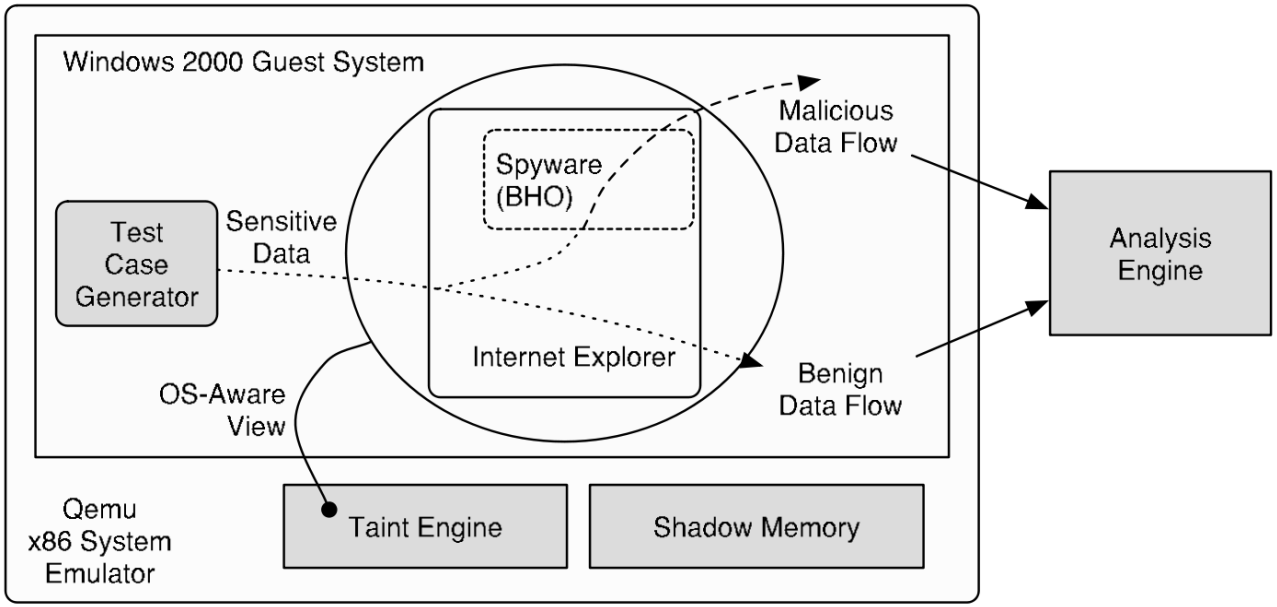


Figure: System overview

Dynamic taint propagation

Taint propagation (data dependencies):

- All **output** bytes of an operation marked as **tainted** whenever:
  - **any input operand** is **tainted**
  - the register containing **the memory address to access** is **tainted**

Shadow memory:

- **Keep track** of the taint status of all bytes in memory (+ GP registers)
- **One byte per byte** of memory
  $\longrightarrow$ allows to track **which** information the data in memory depends on

Dynamic taint propagation

Taint propagation (data dependencies):

- All **output** bytes of an operation marked as **tainted** whenever:
  - **any input operand** is **tainted**
  - the register containing **the memory address to access** is **tainted**

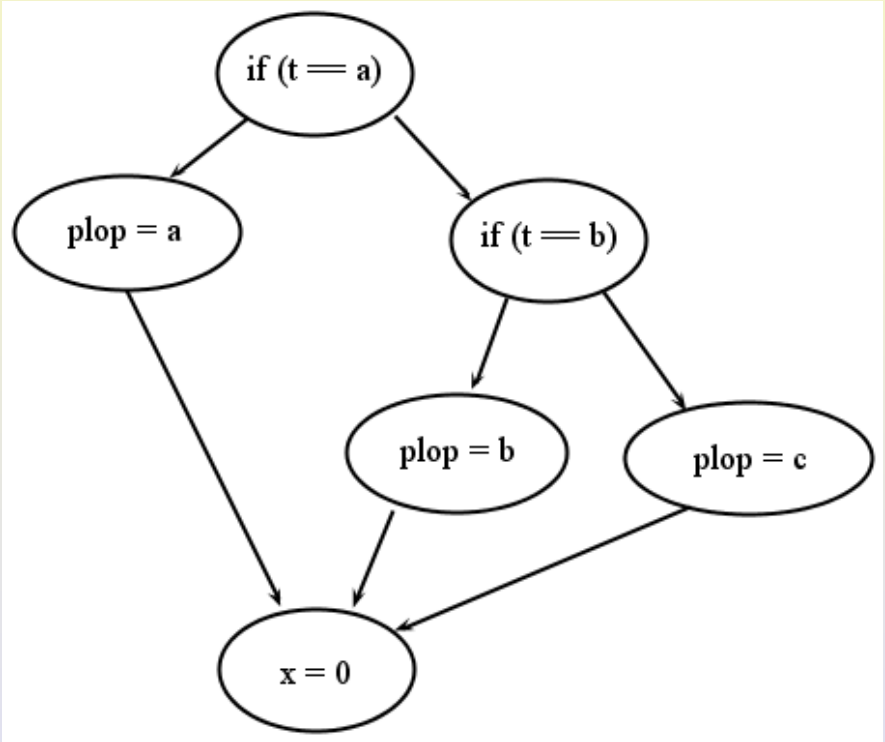
Shadow memory:

- **Keep track** of the taint status of all bytes in memory (+ GP registers)
- **One byte per byte** of memory
  $\longrightarrow$ allows to track **which** information the data in memory depends on

However, detection is easy to evade... Can someone see how?

Evading detection of data dependencies

Consider that `t` contains some **sensitive data**:



The is no **direct** data dependency between `t` and `plop`
$\longrightarrow$ value `plop` will **not** be tainted

Solution: Direct Control Dependencies

Solution:

- Handle **control dependencies**
- If the **execution** of an instruction **depends on a tainted variable**
  $\longrightarrow$ the **output** of the instruction is **tainted**

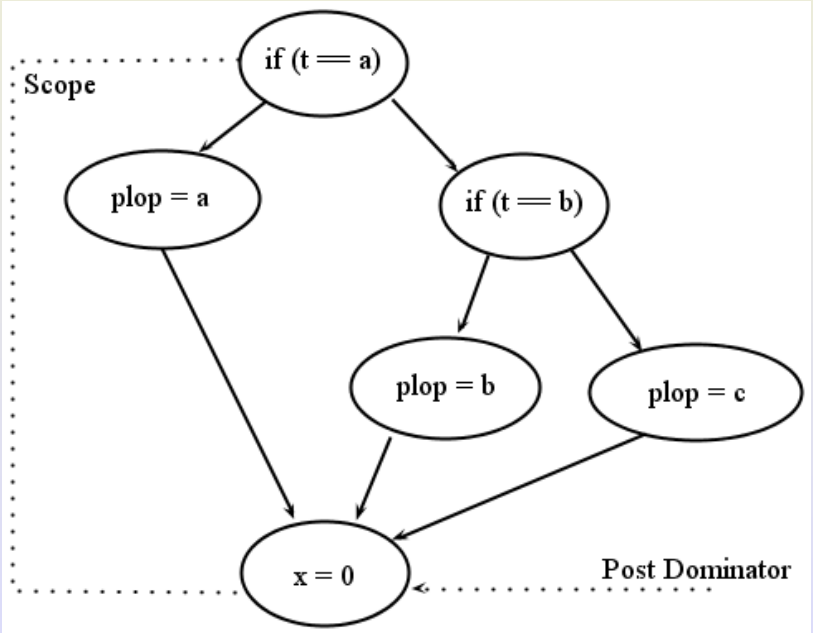
Direct Control Dependencies Handling

To handle **control dependencies**:

- The taint engine examines all encountered **conditional branch instructions** during execution
- If the **branch condition** depends on a tainted value
  $\longrightarrow$ identify all instructions **conditionally dependent** on the result of the branch, using static analysis

Post-dominator

- Find the **post-dominators** of the branch instruction. (i.e. the instructions that will be executed whether the condition is true or false)
- Continue the execution, and **taint all outputs** until a post-dominator is reached



Untainting

Untainting:

- Tainting is erased when the register or memory location is overwritten:
  - `movb %eax, 0x8` : constants untainted
- Can be tricky on x86:
  - `xor %eax, %eax` : zeroing %eax
  - Others cryptic ways to zero a value (with subs, etc...)

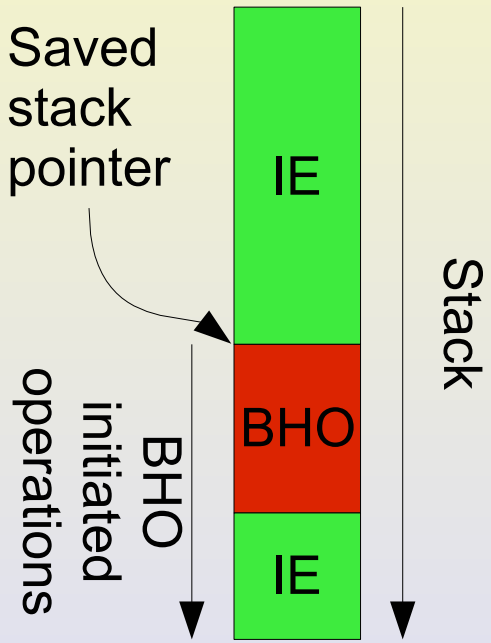
Operating System Awareness

- **Whole operating system** running
- Need to **identify when IE is active**
  $\longrightarrow$ **Hooks** in operating system
  $\longrightarrow$ Uses **page table physical address** as an identifier
- And **where BHO is loaded** in IE address space

Operating System Awareness - BHO

To know if an operation is **initiated by BHO**:

- When execution passes from IE to BHO
  → **save the stack pointer**.
- As long as the stack pointer is **below** that value → **flow initiated by BHO**.
- These outputs marked as **"suspicious"**
  → if data is **leaked**
  → **malicious flow**



Performance

- Static analysis used **only in BHO** (IE assumed sane)
- Avoids **taint label explosion**
- Still relatively slow, but **can be used by analysts**

	Min	Max	Average
Native Windows	0.6	2.9	1.9
QEMU	1.8	6.1	3.6
Modified QEMU	17.3	79.4	35.7

Table: Page loading times

Evaluation

Various **spywares/benign BHO** tested:

- **All spyware** correctly classified
- **All but 2 benign BHO** correctly classified

	Spyware	FN	Benign	Suspicious	FP
Spyware	21	0	-	-	-
Benign	-	-	12	1	1

Table: Spyware/benign BHO analysis

Evaluation - False positive

`PrivacyBird` BHO:

- Privacy management standard implementation.
- Retrieves a privacy policy (`http://website/w3c/p3p.xml`)

Evaluation - False positive

`PrivacyBird` BHO:

- Privacy management standard implementation.
- Retrieves a privacy policy (`http://website/w3c/p3p.xml`)

Example:

- When `http://www.epfl.ch/` is accessed.
- BHO accesses `http://www.epfl.ch/w3c/p3p.xml`

Evaluation - False positive

`PrivacyBird` BHO:

- Privacy management standard implementation.
- Retrieves a privacy policy (`http://website/w3c/p3p.xml`)

Example:

- When `http://www.epfl.ch/` is accessed.
- BHO accesses `http://www.epfl.ch/w3c/p3p.xml`
- URL is leaked!

Evaluation - False positive

`PrivacyBird` BHO:

- Privacy management standard implementation.
- Retrieves a privacy policy (`http://website/w3c/p3p.xml`)

Example:

- When `http://www.epfl.ch/` is accessed.
- BHO accesses `http://www.epfl.ch/w3c/p3p.xml`
- URL is leaked!

But the information is not sent to the attacker!
→ **false positive**

Evaluation - Suspicious Sample

`LostGoggles` BHO:

- Adds images to Google search
- **Downloads** a JavaScript locally, then embeds it in the Google search

Evaluation - Suspicious Sample

`LostGoggles` BHO:

- Adds images to Google search
- **Downloads** a JavaScript locally, then embeds it in the Google search

Problem:

- During the **first access** to Google, downloads the JavaScript.
- In the request, the `Referer` field is set!

HTTP Request:

```
GET /script/LostGoggles.js HTTP/1.1
...
Referer:  http://www.google.ch/search?q=SAV+what+a+fun+course
```

Evaluation - Suspicious Sample

`LostGoggles` BHO:

- Adds images to Google search
- **Downloads** a JavaScript locally, then embeds it in the Google search

Problem:

- During the **first access** to Google, downloads the JavaScript.
- In the request, the `Referer` field is set!
  → **information leak** (probably unintended).

HTTP Request:

```
GET /script/LostGoggles.js HTTP/1.1
...
Referer:  http://www.google.ch/search?q=SAV+what+a+fun+course
```

Conclusion

- **Dynamic spyware analysis**
- Switch to **static analysis for control dependencies**
- **Detailed** reports about **which** sensitive data is leaked and **where** it is sent

Thanks for your attention

Questions ?