Brian Johannesmeyer, Asia Slowinska, Herbert Bos, & Cristiano Giuffrida

VUSec

Practical Data-Only Attack Generation

Published at ***Usenix Security 2024*** and ***;login: Magazine***

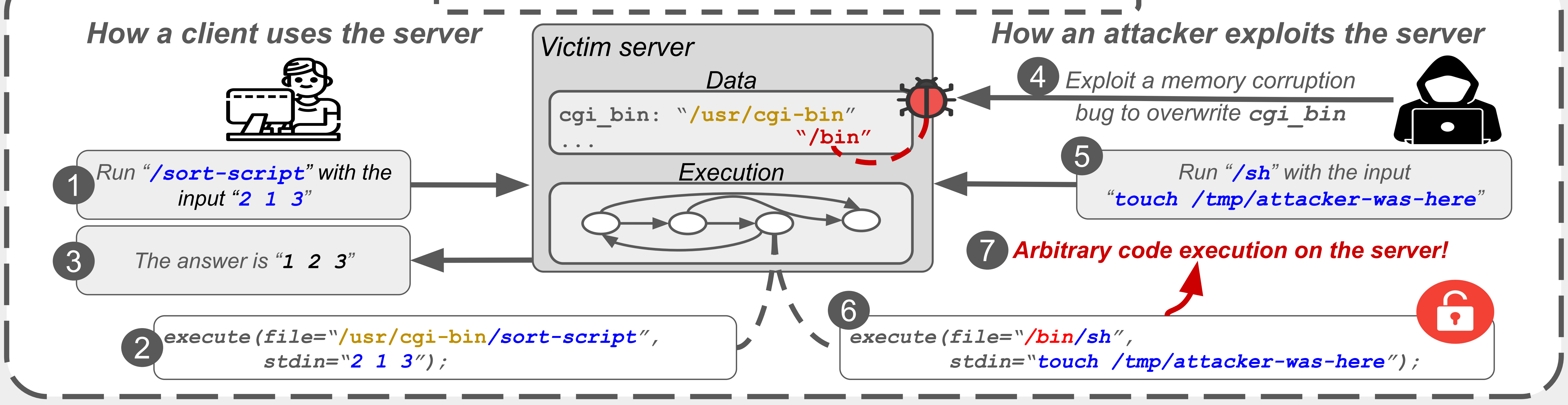
Background

Definition

*Data-only attacks do not corrupt a victim's **control flow**.*

*Instead, they corrupt other data, e.g., **function arguments**.*

Example data-only attack



How a client uses the server

1 Run "`/sort-script`" with the input "`2 1 3`"

2 `execute(file="/usr/cgi-bin/sort-script", stdin="2 1 3");`

3 The answer is "`1 2 3`"

Victim server

Data

`cgi_bin: "/usr/cgi-bin"` `"/bin"`
`...`

Execution

How an attacker exploits the server

4 Exploit a memory corruption bug to overwrite `cgi_bin`

5 Run "`/sh`" with the input "`touch /tmp/attacker-was-here`"

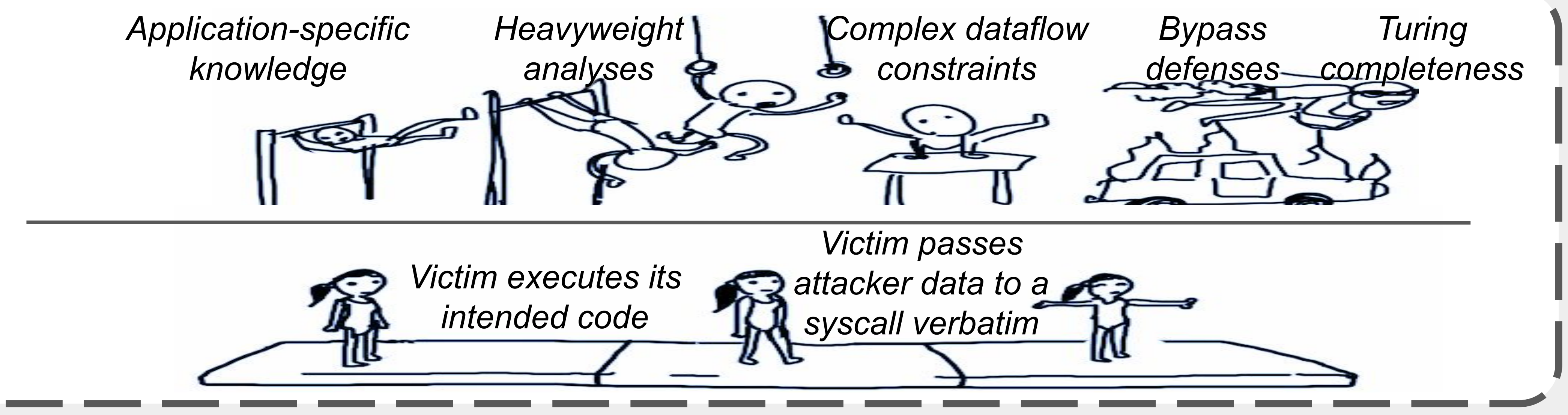
6 `execute(file="/bin/sh", stdin="touch /tmp/attacker-was-here");`

7 ***Arbitrary code execution on the server!***

Building data-only attacks with Einstein

*Data-only attacks are **not considered a practical threat** because they may require:*

*We can **easily build data-only attacks** by leveraging insights from the example attack:*



Application-specific knowledge

Heavyweight analyses

Complex dataflow constraints

Bypass defenses

Turing completeness

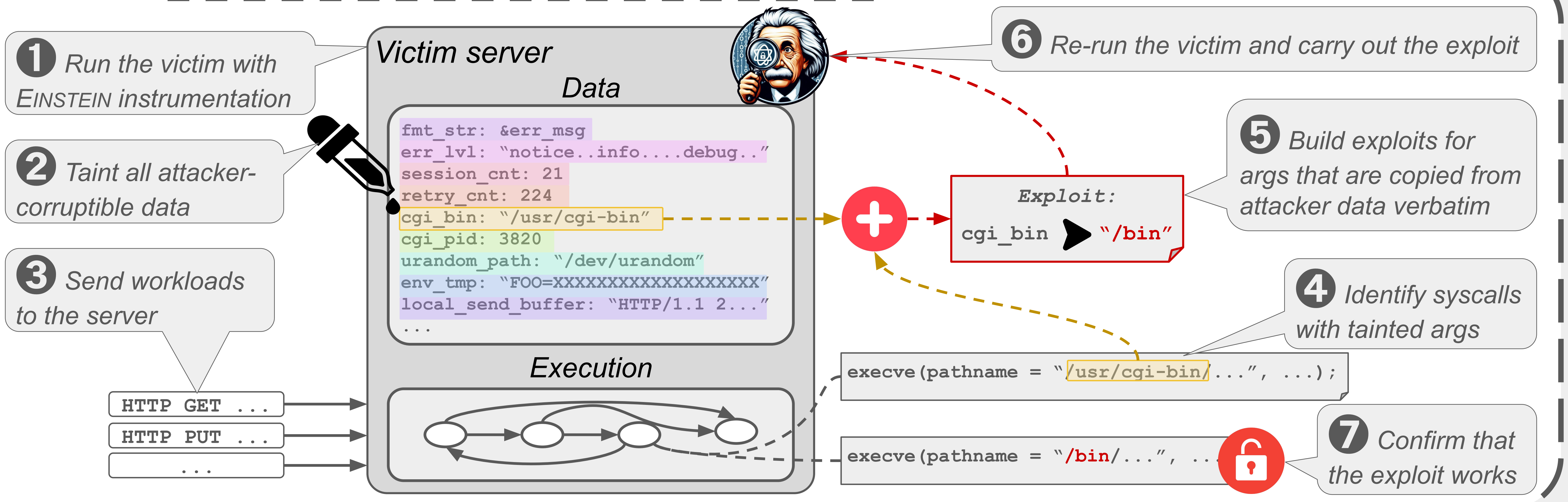
Victim executes its intended code

Victim passes attacker data to a syscall verbatim

Building the example attack

*Determine **which** data to overwrite using **dynamic taint analysis***

*Determine **what** to overwrite it with by targeting data copied **verbatim***



1 Run the victim with Einstein instrumentation

2 Taint all attacker-corruptible data

3 Send workloads to the server

```
HTTP GET ...
HTTP PUT ...
...
```

Victim server

Data

```
fmt_str: &err_msg
err_lvl: "notice..info....debug.."
session_cnt: 21
retry_cnt: 224
cgi_bin: "/usr/cgi-bin"
cgi_pid: 3820
urandom_path: "/dev/urandom"
env_tmp: "FOO=XXXXXXXXXXXXXXXXXXXX"
local_send_buffer: "HTTP/1.1 2..."
...
```

Execution

4 Identify syscalls with tainted args

`execve(pathname = "/usr/cgi-bin/...", ...);`

5 Build exploits for args that are copied from attacker data verbatim

Exploit:
`cgi_bin` ➤ `"/bin"`

6 Re-run the victim and carry out the exploit

`execve(pathname = "/bin/...", ...`

7 Confirm that the exploit works

Results

Attacks generated for `nginx`

Vulnerable ***`execve`*** → Code-Execution

*Vulnerable file-configuring syscall (e.g., **`openat`**) + Vulnerable file-write syscall (e.g., **write**)* → Write-What-Where

*Vulnerable socket-configuring syscall (e.g., **`connect`**) + Vulnerable socket-write syscall (e.g., **sendmsg**)* → Send-What-Where

Attack primitive	Count
Code-Execution	1
Write-What-Where	17
Write-What	375
Write-Where	79
Send-What-Where	41
Send-What	372
Send-Where	59
Total	**944**

*We identify **thousands of vulnerable gadgets** in popular web servers.*

*We present two case studies of attacks that **bypass state-of-the-art defenses**.*

*Our data-only attacks call upon researchers and vendors to **rethink mitigation strategies**.*