



Sandia National Laboratories

CHIRP

Cloud Hypervisor Forensics and Incident Response Platform

Vince Urias

SANDIA NATIONAL LABORATORIES
DOE PACT VIRTUAL SHOWCASE

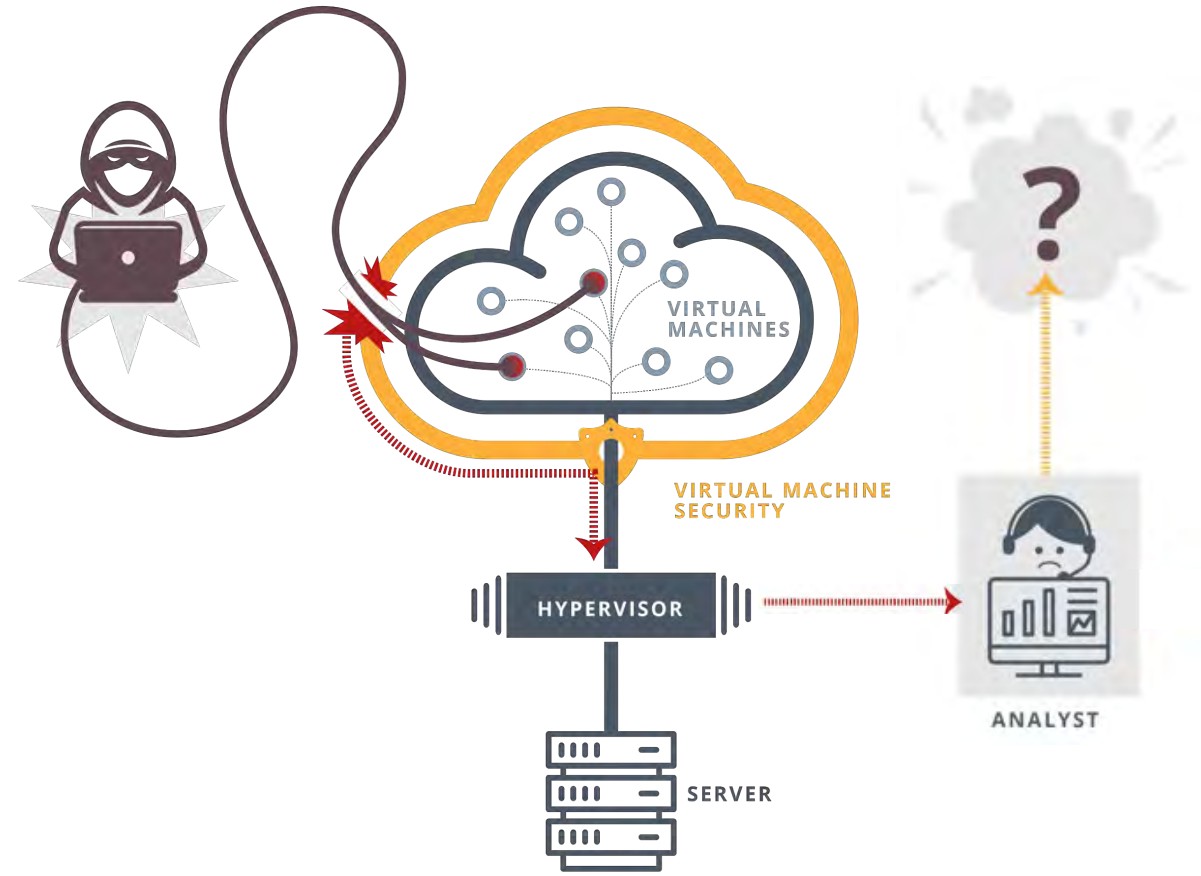
SAND2020-6433 PE



Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

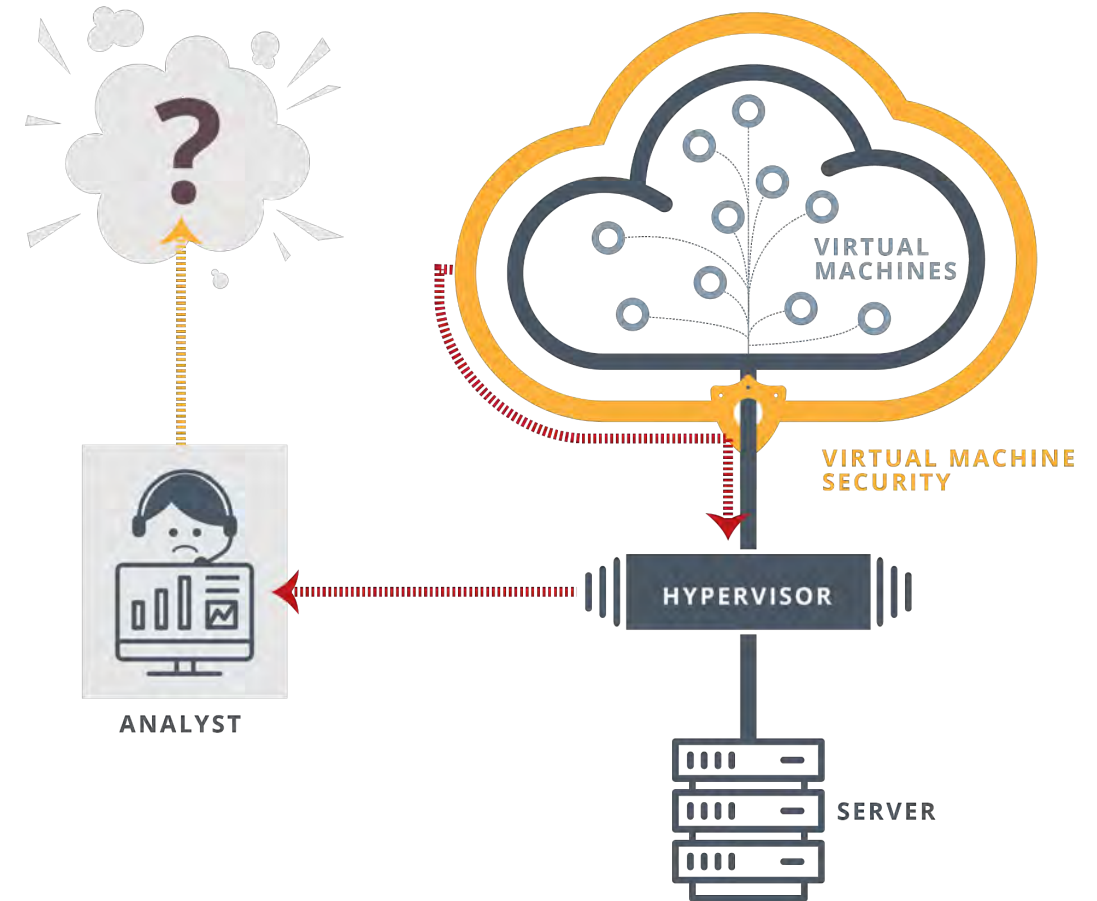


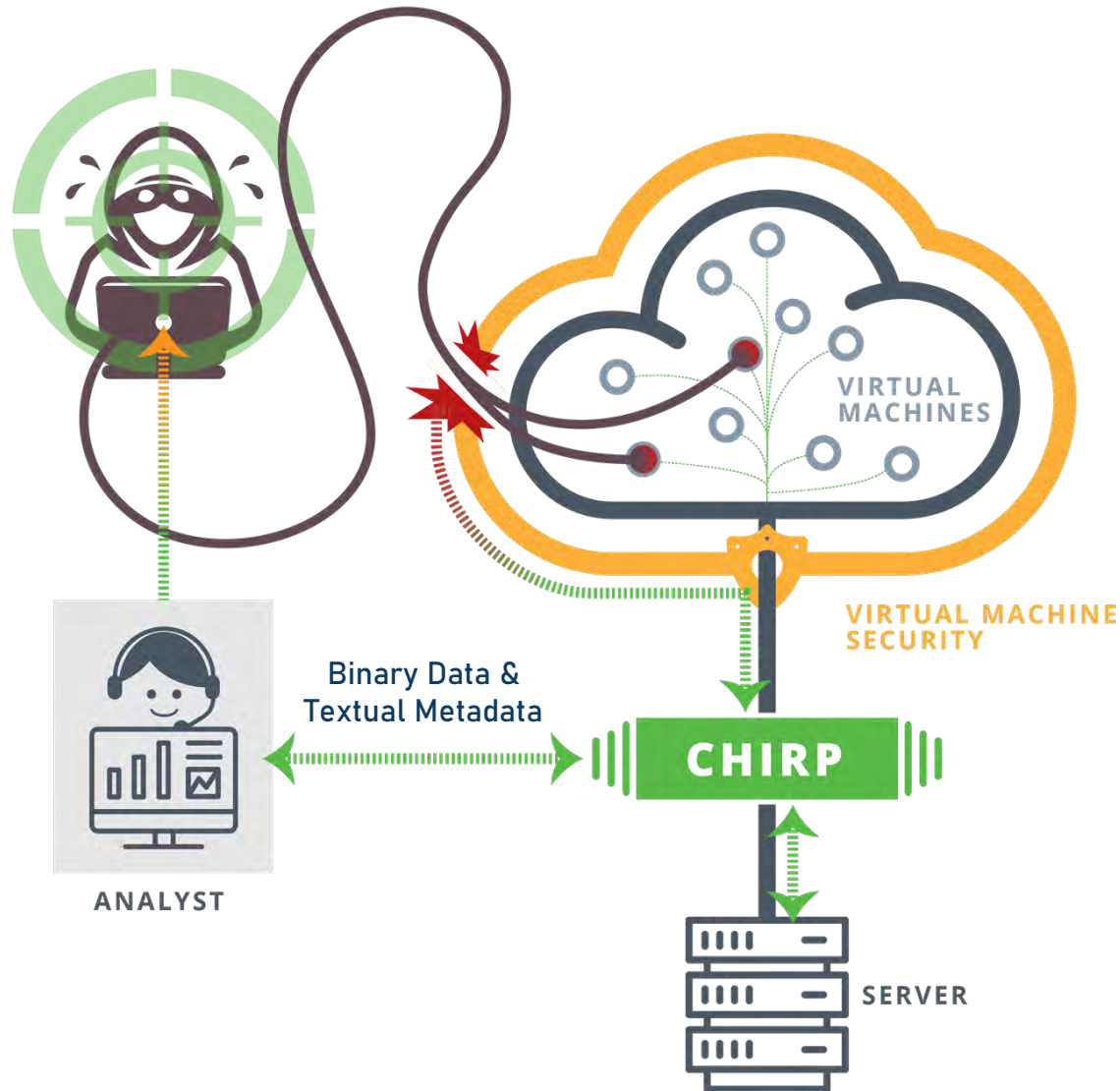
When cyber incidents occur in the cloud, the SOC Analyst has no visibility beyond the hypervisor.





- Ephemeral nature of cloud environment
- Hypervisor instrumentation with **minimal** overhead
- Hypervisor agnostic
- OS agnostic
- Common data model
- Broad data set collection





Virtual Machine Introspection (VMI)

- Sandia has developed a custom VMI implementation to address the issue of overhead.
- Platform agnostic
- Text and binary data collection
- Allows correlation with other sources



Principal Investigator Vince Urias

- 20 Years of Cyber experience at Sandia Labs
- Numerous national awards

Developmental History

2016 Initial Development starts

2017 First disclosed (Dec)

2019 R&D 100 Winner (May)

Funding History

~\$1M LDRD

Research Team

William Stout

Caleb Loverro

Technology Readiness Level (TRL):
TRL 6 as of June 2020

IP Protection

- Patent Application # 16/051,005 filed July 31, 2018
- Copyright approved for commercial licensing – May 2018

Market Validation

2 government deployments





TECHNICAL REQUIREMENTS

Access to the Hypervisor

Designed for IaaS from the start

Off prem and on prem cloud-ready

BENEFITS

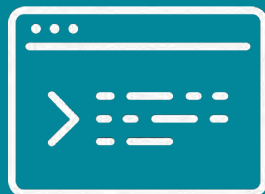
Lightweight

Real-time dynamic response

Configurable logging → incident response or forensics



Management
Dashboard



Expanded
Analytics



Improve
Deployment



Rigorous
Documentation



Private sector
pilot for market
validation

Bringing CHIRP to Market

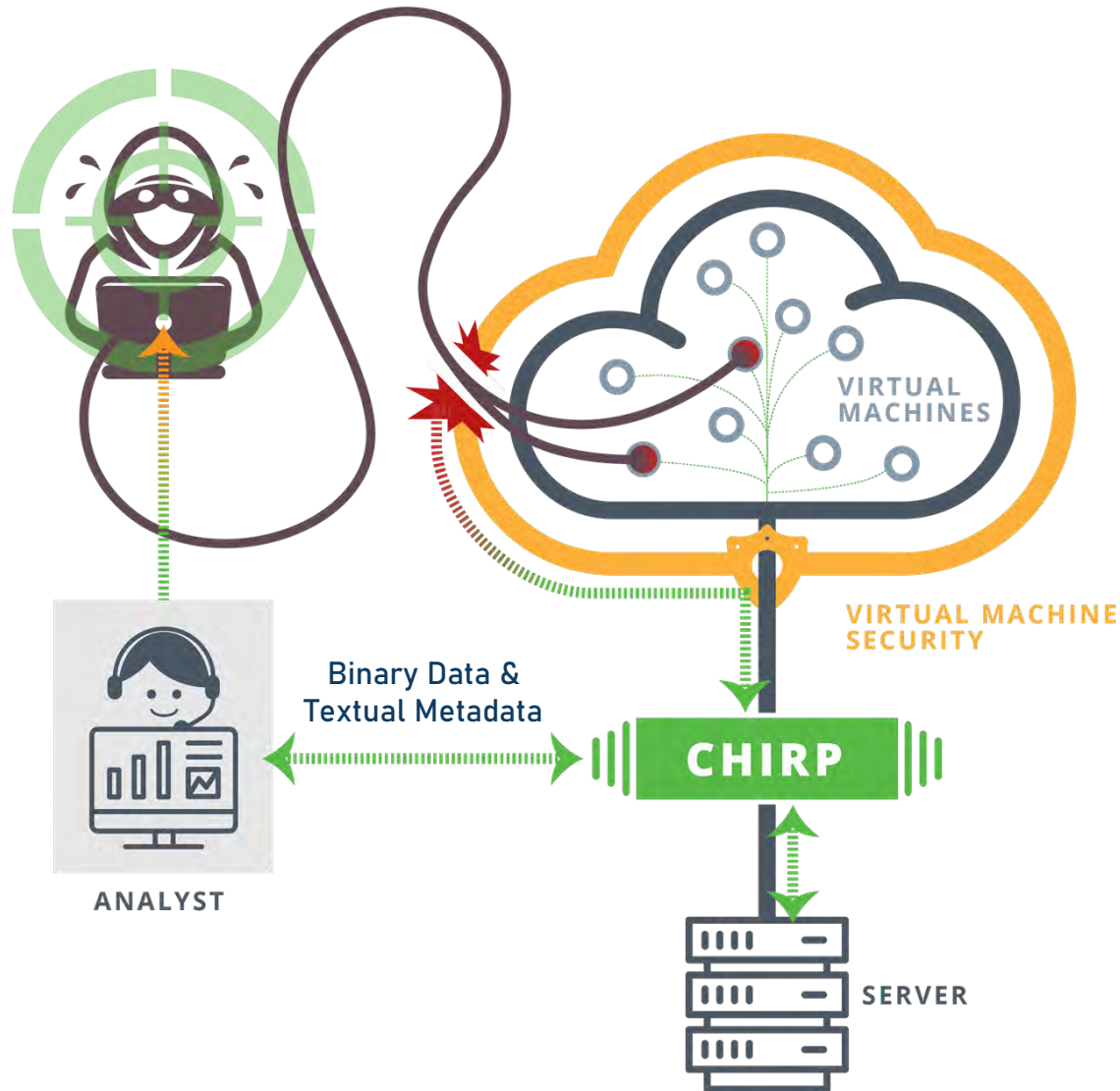


Sandia National Laboratories

CHIRP

Cloud Hypervisor Forensics and Incident Response Platform

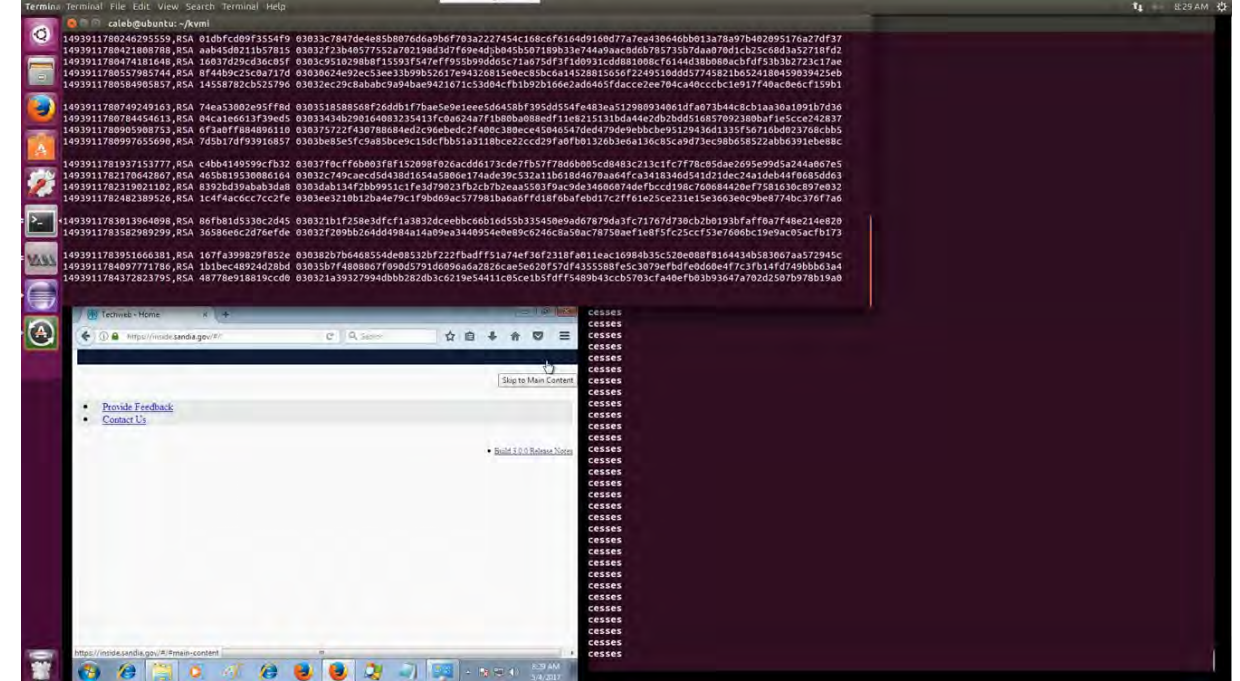
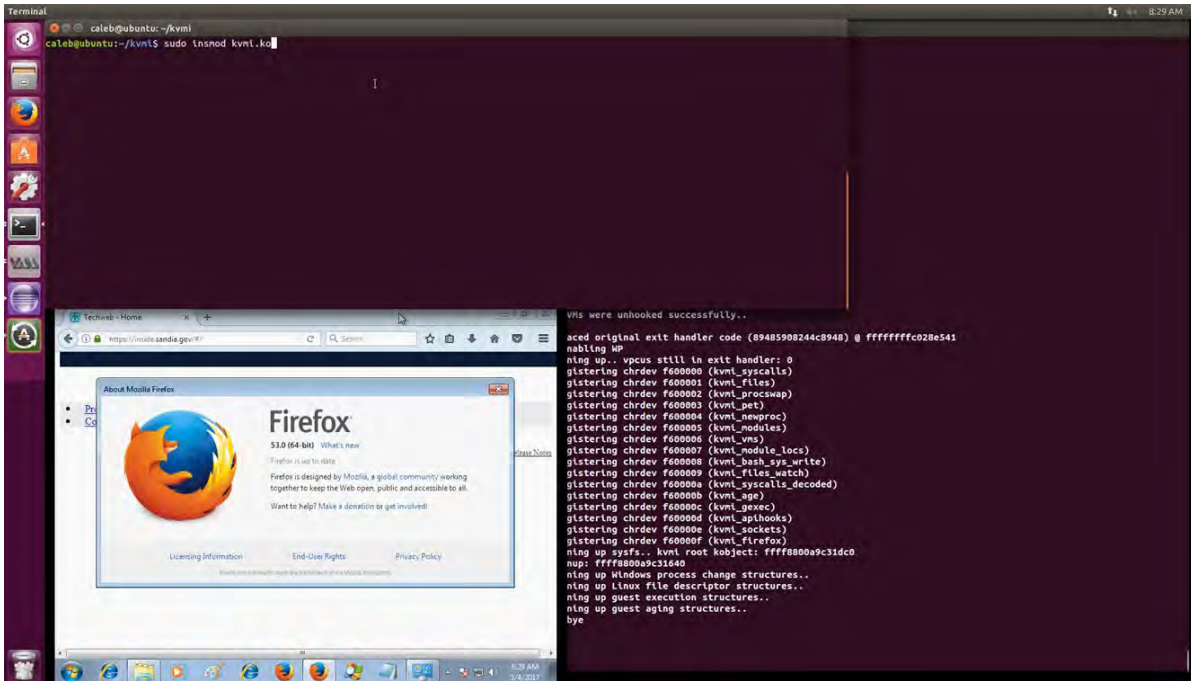
DOE PACT VIRTUAL SHOWCASE
DEMONSTRATION



Vignette Demonstrations

- Module Dumping
- Key Extraction
- Shell
- Carving
- System Call Decoding
- Reverse Engineering Pipeline
- Big Data Extraction

Key Extraction



Keys used for encryption are extracted from processes (e.g., Firefox, Windows LSASS, etc.) Here, Firefox browses to sandia.gov, RSA keys for the session are extracted.

Able to see within the session... eliminating man-in-the-middle attacks.



```
root@eadevnode:~  
4081      0      0 kworker/0:2  
4088      0      0 bash  
4106      0      0 kworker/u2:2  
4111      0      0 kworker/u2:0  
4113      0      0 kworker/u2:1  
4117      0      0 vi  
28660     0      0 xfsalloc  
28666     0      0 xfs_mru_cache  
28673     0      0 jfsIO  
28674     0      0 jfsCommit  
28675     0      0 jfsSync  
kvmi[0]$ pscarve vi  
kvmi[0]$  
Thank you for using KVM!  
root@eadevnode:~# ls  
btrfs      iso      kvmi.ko      linuxdumpprocs.py  shell.py  vms  
dumpmodules.py  kernel  linuxdumpmodules.py  scripts   vi_4117  
root@eadevnode:~# ls btrfs/  
btrfs metadata  
root@eadevnode:~# cat btrfs/metadata  
module: btrfs  
init: 0xffffffffc020698b  
exit: 0xffffffffc01c6f21  
num_syms: 0  
guest virtual address: 0xffffffffc0113000  
size: 0xf1000  
text size: 0xb4000  
read-only size: 0xc8000  
root@eadevnode:~# ls kernel/  
memory metadata  
root@eadevnode:~# ls vi_4117/  
0 1 2 3  
root@eadevnode:~# █
```

```
VNC: QEMU  
  
VIM - Vi IMproved  
version 7.4.1689  
by Bram Moolenaar et al.  
Modified by pkg-vim-maintainers@lists.alioth.debian.org  
Vim is open source and freely distributable  
  
Become a registered Vim user!  
type :help register<Enter> for information  
  
type :q<Enter> to exit  
type :help<Enter> or (F1) for on-line help  
type :help version?<Enter> for version info  
  
0,0-1 All
```

This example shows how the analyst can carve the VI text editor application (process) from the VM

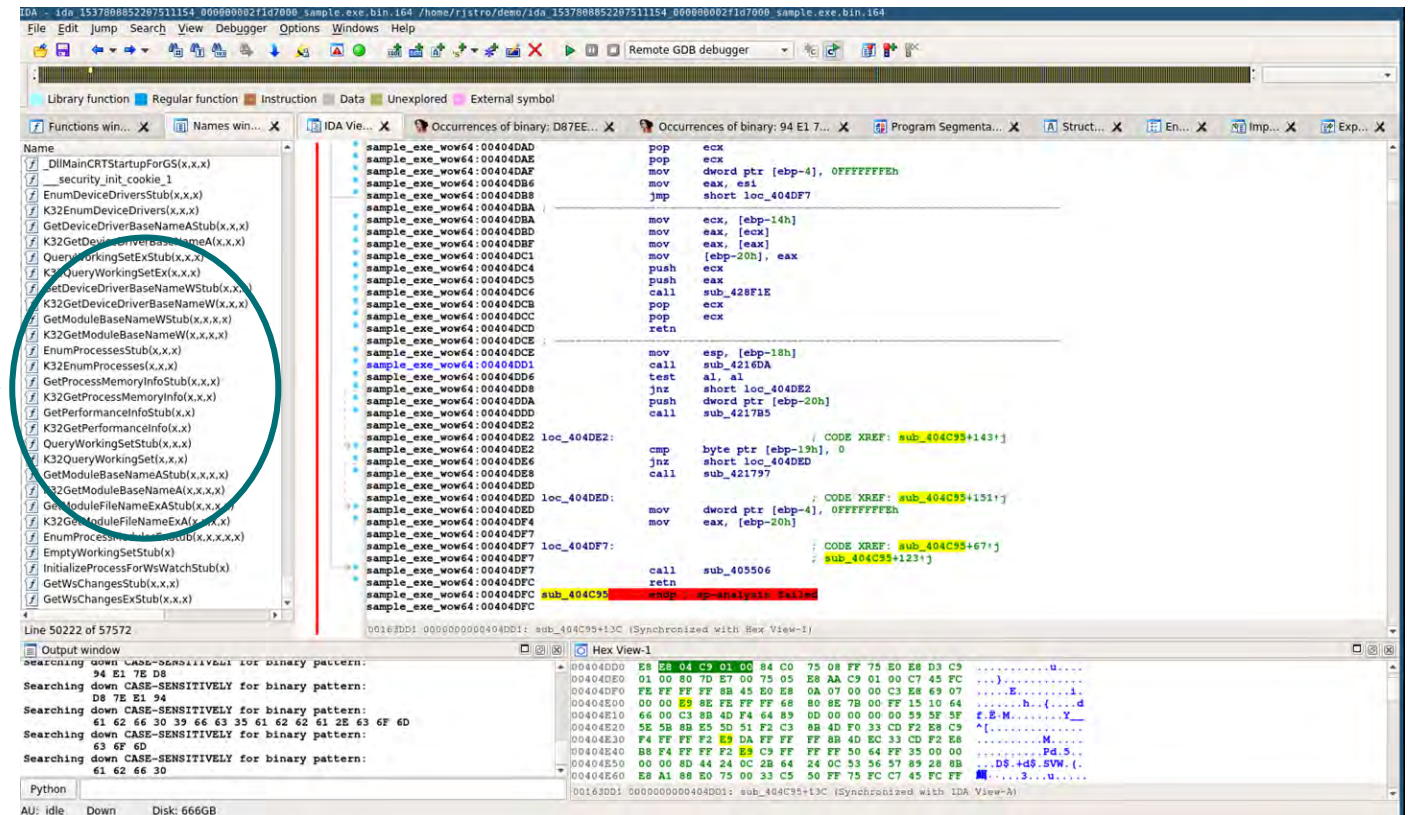
Reverse Engineering Pipeline

CHIRP
DEMONSTRATION



- Track all memory for a process, including all libraries and OS code (not just the binary itself).
- Copy of all these mappings as binary that can be loaded into IDA, with multiple copies so program state can be analyzed at different times during execution.
- An idapython script is used to relocate all those mappings to correct locations, providing the reverse engineer deep insight into a given process.

Automatically gather symbols and library information, and correctly relocate (in memory)



Big Data Extraction



Feature	Linux	Windows	Mac
Raw syscalls	Y	Y	Y
Decoded syscalls	Y	Y	Y
PID/proc name extraction	Y	Y	Y
Guest execution	Y	Y	N
Kernel carving	Y	Y	N
Module carving	Y	Y	N
Process carving	Y	Y	N
Process tracking (start & exit)	Y	Y	N
File extraction	Y	Y	N
Biometrics	Y	N	N
Socket chardev	N	Y	N

Registry Key Access

src_process	handle	registry_key	ret
sample.exe	184	Extensible Cache	STATUS_SUCCESS:0x0
sample.exe	428	Software\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\Cache	STATUS_SUCCESS:0x0
sample.exe	428	Extensible Cache	STATUS_SUCCESS:0x0
sample.exe	184	Software\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\Cache	STATUS_SUCCESS:0x0
sample.exe	184	Extensible Cache	STATUS_SUCCESS:0x0
sample.exe	428	Software\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\Cache	STATUS_SUCCESS:0x0
sample.exe	428	Extensible Cache	STATUS_SUCCESS:0x0
sample.exe	184	Software\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\Cache	STATUS_SUCCESS:0x0
sample.exe	184	Software\Microsoft\Windows\CurrentVersion\PeerDist\Service	STATUS_SUCCESS:0x0
sample.exe	184	REGISTRY\MACHINE\SOFTWARE\Wow6432Node\Piriform\Agomo	STATUS_SUCCESS:0x0
sample.exe	184	{91150000-0011-0000-1000-00000000FFICE}	STATUS_SUCCESS:0x0
sample.exe	184	{90150000-0120-0409-1000-00000000FFICE}	STATUS_SUCCESS:0x0
sample.exe	184	{90150000-0117-0409-1000-00000000FFICE}	STATUS_SUCCESS:0x0
sample.exe	184	{90150000-0115-0409-1000-00000000FFICE}	STATUS_SUCCESS:0x0
sample.exe	184	{90150000-00E2-0409-1000-00000000FFICE}	STATUS_SUCCESS:0x0
sample.exe	184	{90150000-00E1-0409-1000-00000000FFICE}	STATUS_SUCCESS:0x0
sample.exe	184	{90150000-00C1-0409-1000-00000000FFICE}	STATUS_SUCCESS:0x0
sample.exe	184	{90150000-00C1-0000-1000-00000000FFICE}	STATUS_SUCCESS:0x0
sample.exe	184	{90150000-00BA-0409-1000-00000000FFICE}	STATUS_SUCCESS:0x0
sample.exe	184	{90150000-00A1-0409-1000-00000000FFICE}	STATUS_SUCCESS:0x0

Page Execution Coverage (Including Libraries)

pages_of_code	pages_touched	percent_coverage
9621	1662	17.27471156844403

Running Processes

pid	wow64	name	user	commandline
504	1	sample.exe	user	C:\Users\user\Downloads\sample.exe
70	0	conhost.exe	SYSTEM	\\?\C:\Windows\system32\conhost.exe
31c	0	cmd.exe	user	cmd /c 'D:\launch.bat'
294	0	rundll32.exe	user	'C:\Windows\system32\RunDLL32.EXE' Shell32.DLL,ShellExec_RunDLL D:\launch.bat
7b0	0	ipconfig.exe	user	ipconfig /renew

Page Execution Coverage (binary only)

total_pages	pages_touched	percent_coverage
821	393	47.868453105968335

Loaded Modules

fullname	size	loadcount	driver
C:\Windows\system32\aelupsvc.dll	15000	1	0
C:\Windows\system32\srvc11.dll	19000	2	0
C:\Windows\system32\netutils.dll	9000	1	0
C:\Windows\system32\netapi32.dll	11000	1	0
C:\Windows\system32\uxtheme.dll	80000	4	0

NtOpenProcess

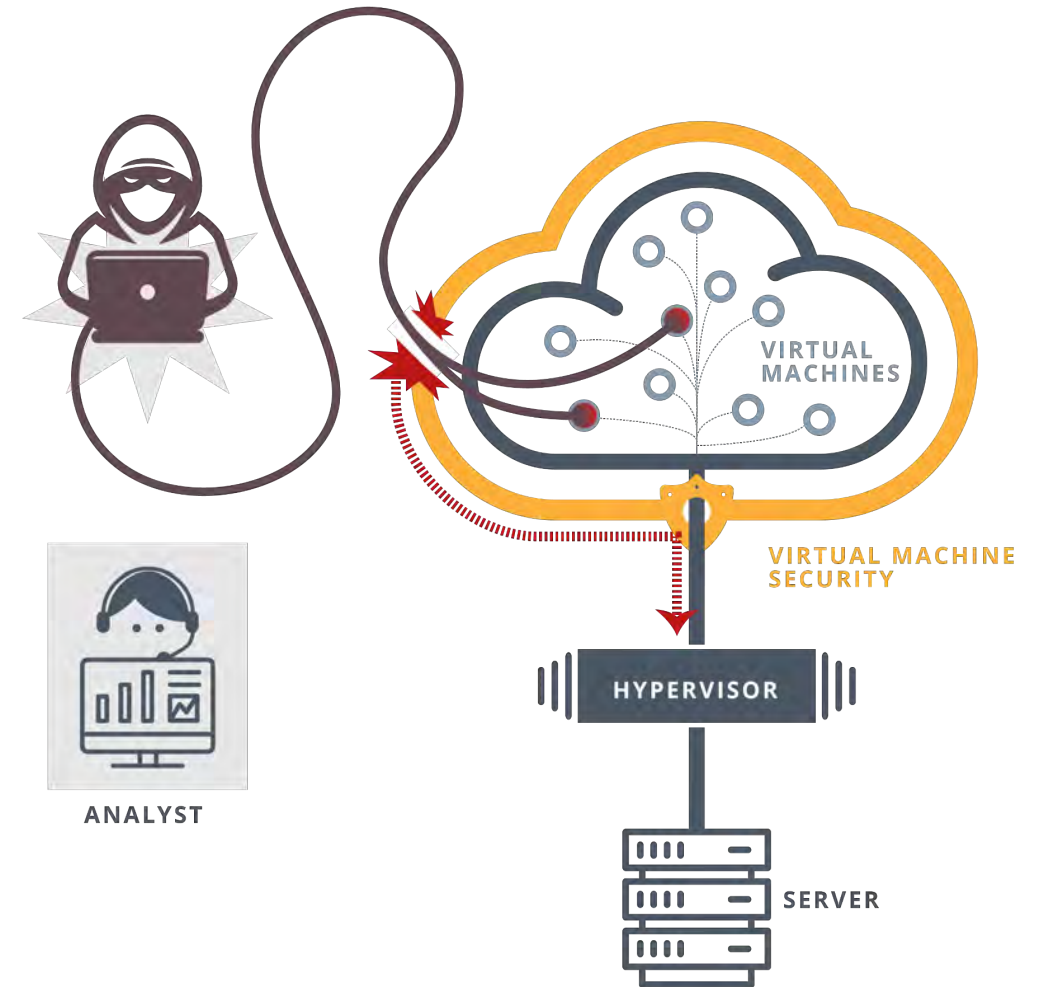
src_process	dst_process	access_mask	ret
sample.exe	conhost.exe	ACCESS_MASK:1000	STATUS_SUCCESS:0x0
sample.exe	cmd.exe	ACCESS_MASK:1000	STATUS_SUCCESS:0x0
sample.exe	GoogleUpdate.exe	ACCESS_MASK:1000	STATUS_SUCCESS:0x0
sample.exe	explorer.exe	ACCESS_MASK:1000	STATUS_SUCCESS:0x0
sample.exe	taskeng.exe	ACCESS_MASK:1000	STATUS_SUCCESS:0x0
sample.exe	taskhost.exe	ACCESS_MASK:1000	STATUS_SUCCESS:0x0
sample.exe	svchost.exe	ACCESS_MASK:1000	STATUS_SUCCESS:0x0
sample.exe	svchost.exe	ACCESS_MASK:1000	STATUS_SUCCESS:0x0
sample.exe	svchost.exe	ACCESS_MASK:1000	STATUS_SUCCESS:0x0
sample.exe	spoolsv.exe	ACCESS_MASK:1000	STATUS_SUCCESS:0x0

Reserve Slides





Real-time forensics and incident response for Cloud Service Providers providing deeper analytics of security incidents—leading to better protection and improved visibility.





Virtual Machine Introspection (VMI)

- Sandia has developed a custom VMI implementation to address the issue of overhead.
- Platform agnostics
- Text and binary data collection
- Allows correlation with other sources

Cloud user accesses the cloud system to instantiate a VM

The CHIRP VMI module is loaded on the Cloud host

The CHIRP intercepts communication between guest and host

Binary data is extracted from the VMI and output to the host

Textual metadata is extracted from the VM and output to the host

Analyst accesses CHIRP to perform duties

CHIRP Process Flow



Cloud user accesses the Cloud system to instantiate a VM

VM specification is sent to a hypervisor on the host.

Hypervisor allocates resources to the guest VM and starts it.

The CHIRP VMI module is loaded on the Cloud host

CHIRP is injected into kernel-space of the host Cloud server

CHIRP detects hypervisor and places itself in the same administrative domain

The CHIRP intercepts communication between guest and host

CHIRP hooks VM-exit handler, assuming hypervisor functions

Dynamically detects all VM Operating Systems and state on the host

Binary data is extracted from the VM and output to the host

Binary data (memory, files, documents, malware, etc.) stored to the host

Binary data are analyzed

Textual metadata is extracted from the VM and output to the host

Textual content (e.g., logs) are ingested by a Security Information and Event Management (SIEM) system

Textual data is correlated against other data (network, system) in the SIEM

Analyst accesses CHIRP to perform duties

Inspects binary data (executes files, reverse engineers malware, etc.)

Inspects textual data to identify indicators of compromise or attack

Modifies VM state as needed to aid/prolong an investigation



Vignette Demonstrations

- Module Dumping
- Key Extraction
- Shell
- Carving
- System Call Decoding
- Reverse Engineering Pipeline
- Big Data Extraction

Interactive Shell



```
root@esatvnode:~# # This is the host
root@esatvnode:~# ./shell.py
Welcome to the KVMi shell. Type 'help' or '?' to list commands.
If this is your first time, run the first_time command.

kvmi$ first_time
The general workflow for the shell is to list the running VMs with the "list"
command.
After doing that, you should "select" a VM to issue commands to, and then issu
e it commands.
kvmi$ help

Documented commands (type help <topic>):
=====
EOF          guest_execute_adv  list_flags        modcarve         select
debug       qvatomhva        list_libs         pause            set_flag
exit        help             list_modules     pscarve          shell
first_time  kernelcarve      list_processes   read_memory     write_memory
guest_execute list              isof              result

kvmi$ list
List of VMs:
(0) pid: 22473, linux
kvmi$ select 0
kvmi[0]$
```

```
ubuntu@ubuntu1604m:~$ # This is the guest
ubuntu@ubuntu1604m:~$
```

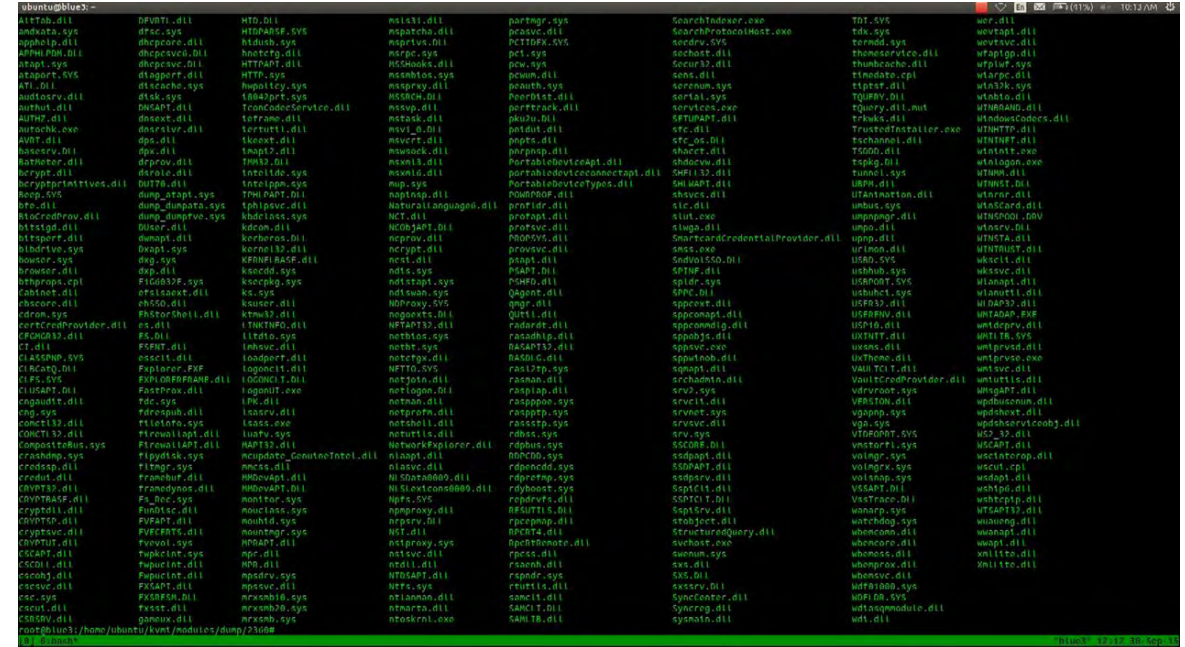
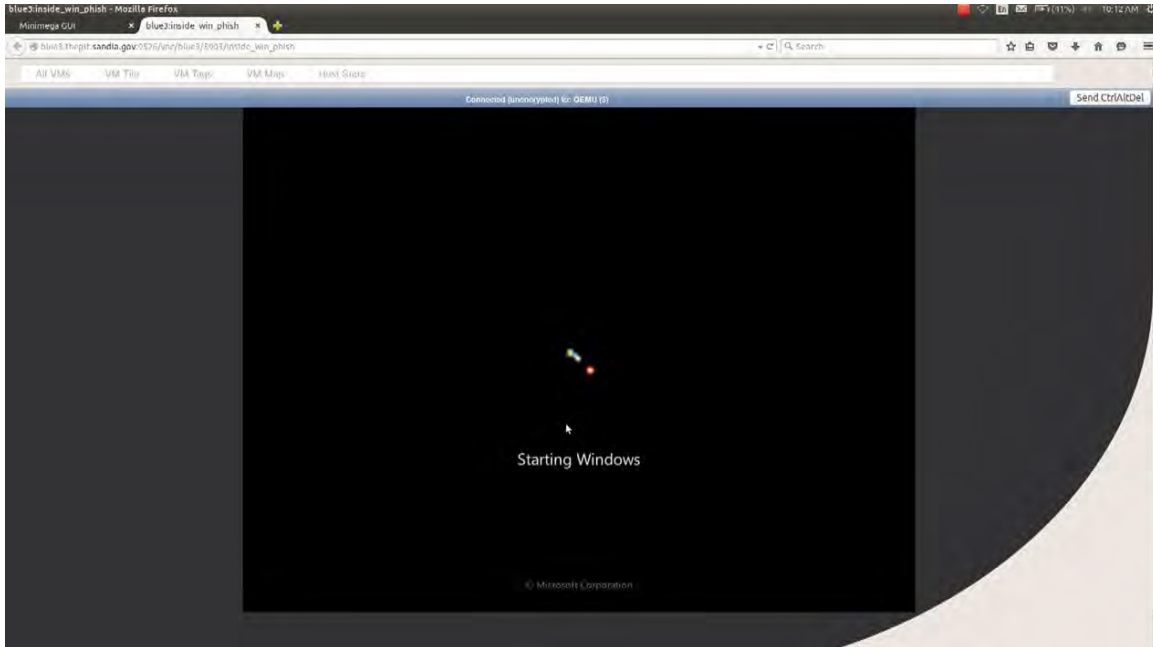
```
lu_cn
ib_cm
ib_sa
ib_mad
ib_core
ib_addr
ib_smi
iscsi_tcp
libiscsi_tcp
libiscsi
scsi_transport_iscsi
autofs4
btrfs
raid10
raid456
async_raid6_recov
async_memcpy
async_pq
async_xor
async_tx
xor
raid6_pq
libcrc32c
raid1
raid0
multipath
linear
hid_generic
usbhid
hid
psmouse
pata_acpi
floppy
```

```
parport          49152  2  ppdev,parport_pc
mac_hid          16384  0
ib_iser          49152  0
ib_iser          49152  1  ib_iser
ib_cm            49056  1  rdma_cm
ib_sa            36864  2  rdma_cm,ib_cm
ib_mad           49152  2  ib_cm,ib_sa
ib_core          106496 6  rdma_cm,ib_cm,ib_sa,lu_cn,ib_mad,ib_iser
ib_smi           20480  2  rdma_cm,ib_core
iscsi_tcp        20480  0
libiscsi_tcp     24576  1  iscsi_tcp
libiscsi         53240  3  libiscsi_tcp,iscsi_tcp,ib_iser
scsi_transport_iscsi  96384  4  iscsi_tcp,ib_iser,libiscsi
autofs4          40960  2
btrfs            987136 0
raid10           49152  0
raid456          110528 0
async_raid6_recov 20480  1  raid456
async_memcpy     16384  2  raid456,async_raid6_recov
async_pq         16384  3  raid456,async_raid6_recov
async_xor        16384  5  async_pq,raid456,async_raid6_recov
async_tx         24576  2  btrfs,async_xor
xor              102400  4  async_pq,raid456,btrfs,async_raid6_recov
raid6_pq         16384  2  xfs,raid456
raid1            36864  0
raid0            20480  0
multipath        16384  0
linear           16384  0
hid_generic      16384  0
usbhid           49152  0
hid              110704 2  hid_generic,usbhid
psmouse         110702 0
pata_acpi        16384  0
floppy           73728  0
ubuntu@ubuntu1604m:~$
```

An interactive shell can be used by the analyst to gather information non-obtrusively from the VM in real-time.

- List VMs
- List libraries in use
- List modules executed
- List file descriptors
- List running processes
- Write into memory/process
- Start processes on the VM
- Carve objects from memory
 - Files
 - Operating system kernel

Module Dumping



Executed binaries are extracted automatically from the guest virtual machine. Here, on initial boot, 500 modules (dynamic linked libraries (*.dll), executables (*.exe, *.sys)) are extracted.

From boot up onward, CHIRP captures valuable information.