Logging to the Danger Zone: Race Condition Attacks and Defenses on System Audit Frameworks

Riccardo Paccagnella, Kevin Liao, Dave Tian, Adam Bates
Logs Are Useful

• 75% of incident response specialists said logs are the most valuable artifact during an investigation.¹

¹ Carbon Black Quarterly Incident Response Threat Report April 2019
75% of incident response specialists said logs are the most valuable artifact during an investigation.¹

CAPEC-268: Audit Log Manipulation

The attacker injects, manipulates, deletes, or forges malicious log entries into the log file, in an attempt to mislead an audit of the log file or cover tracks of an attack.

Hackers are increasingly destroying logs to hide attacks

According to a new report, 72 percent of incident response specialists have come across hacks where attackers have destroyed logs to hide their tracks.

¹ Carbon Black Quarterly Incident Response Threat Report April 2019
Can We Protect the Logs?
Can We Protect the Logs?

• Secure Logging!
Can We Protect the Logs?

- Secure Logging!
- Logs recorded prior to full system compromise cannot be undetectably tampered with.
How Secure Logging Works
How Secure Logging Works

Application \[\rightarrow\] Logger \[\rightarrow\] Logs
How Secure Logging Works

Logging to the Danger Zone: Race Condition Attacks and Defenses on System Audit Frameworks – Riccardo Paccagnella
How Secure Logging Works

1. Application
2. Log filter
3. Syscall processing
4. Logger
5. Kernel log buffer
6. Logs

syscall

User space

Kernel
How Secure Logging Works

Application

Log filter

Syscall processing

Logger

Logs

syscall

User space

Kernel

syscall

syscall 1

kernel log buffer
How Secure Logging Works

Application

Log filter

Syscall processing

syscall

syscall 1

syscall 2

Logger

Logs

 syscall

Kernel

User space

kernel log buffer
How Secure Logging Works

Logging to the Danger Zone: Race Condition Attacks and Defenses on System Audit Frameworks – Riccardo Paccagnella
How Secure Logging Works

This process takes 25-250 ms
How Secure Logging Works

- Logging is asynchronous.
How Secure Logging Works

- Logging is asynchronous.
- It takes 25-250 ms for an event to be logged.
How Secure Logging Works

- Logging is asynchronous.
- It takes 25-250 ms for an event to be logged.

Logs are vulnerable when in the kernel log buffer!
Attack Timeline

Attack begins

$t_1$
Attack Timeline

Attack-related syscall $x_i$ executes

Attack begins

$\bullet t_1$

$\bullet t_i$

$\mathcal{m}_i$
Attack Timeline

- Attack begins
- Attack-related syscall $x_i$ executes
- Event $m_i$ is recorded

$m_i$ is in the kernel buffer (25-250 ms)
Attack Timeline

- Attack begins at time $t_1$
- Attack-related syscall $x_i$ executes at time $t_i$
- Attacker gains root access at time $t_n$
- Event $m_i$ is recorded

$m_i$ is in the kernel buffer (25-250 ms)
Attacker has kernel access ($m_i$ is vulnerable)
Race Condition Attack

Logging to the Danger Zone: Race Condition Attacks and Defenses on System Audit Frameworks – Riccardo Paccagnella

23
Race Condition Attack

Logging to the Danger Zone: Race Condition Attacks and Defenses on System Audit Frameworks – Riccardo Paccagnella
Race Condition Attack

Logging to the Danger Zone: Race Condition Attacks and Defenses on System Audit Frameworks – Riccardo Paccagnella
Realistic Attack Scenario
Realistic Attack Scenario

1. Remote code execution: CVE-2014-6271 (Shellshock)
2. Privilege escalation: CVE-2017-16995
3. Log tampering: log-interceptor (details in the paper)
Realistic Attack Scenario

1. Remote code execution: CVE-2014-6271 (Shellshock)
2. Privilege escalation: CVE-2017-16995
3. Log tampering: log-interceptor (details in the paper)

How many of the 176 log events describing the attack steps above get intercepted?
Realistic Attack Scenario

1. Remote code execution: CVE-2014-6271 (Shellshock)
2. Privilege escalation: CVE-2017-16995
3. Log tampering: log-interceptor (details in the paper)

How many of the 176 log events describing the attack steps above get intercepted?
How do we defend against this attack?
Defense Goals

- **Synchronous integrity**: securing logs synchronously with their creation.
Ways to achieve synchronous integrity

- Attack begins
- Attack-related syscall $x_i$ executes
- Event $m_i$ is recorded
- $m_i$ is in the kernel buffer (25-250 ms)
- Attacker gains root access
- Attacker has kernel access ($m_i$ is vulnerable)

Logging to the Danger Zone: Race Condition Attacks and Defenses on System Audit Frameworks – Riccardo Paccagnella
Ways to achieve synchronous integrity

Option A

Attack begins

Attack-related syscall $x_i$ executes

$m_i$ is in the kernel buffer (25-250 ms)

Attacker gains root access

$m_i$ is recorded

$t_1$

$t_i$

$t_n$

attacker has kernel access

($m_i$ is vulnerable)
Ways to achieve synchronous integrity

Option B

- Attack-related syscall \( x_i \) executes
- Attack begins
- Attack begins at \( t_1 \)
- \( m_i \) is recorded
- Event \( m_i \) is recorded
- \( m_i \) is in the kernel buffer (25-250 ms)
- Attacker gains root access
- Attacker gains root access at \( t_n \)
- Attacker has kernel access (\( m_i \) is vulnerable)

Option A

- Attacker gains root access
- Attacker gains root access at \( t_n \)
KennyLoggings

Attack-related syscall $x_i$ executes

Attack begins $t_1$

$\vdash m_i$

Attacker gains root access $t_n$

Event $m_i$ is recorded

Log events become tamper-evident when they are created
KennyLoggings

Attack-related syscall $x_i$ executes

Attack begins

$\text{Attack begins}$

$\text{Log events become tamper-evident when they are created}$

$\text{Event } m_i \text{ is recorded}$

$\text{Attacker gains root access}$
Log events become tamper-evident when they are created

1. Application executes system call $x_i$
2. Syscall $x_i$ is executed
3. Log event $m_i$ is generated
4. An integrity proof is appended to $m_i$
5. $m_i$ is enqueued to the logging buffer
Log events become tamper-evident when they are created

\[\begin{align*}
1. & \text{Application executes system call } x_i \\
2. & \text{Syscall } x_i \text{ is executed} \\
3. & \text{Log event } m_i \text{ is generated} \\
4. & \text{An integrity proof is appended to } m_i \\
5. & m_i \text{ is enqueued to the logging buffer}
\end{align*}\]
Forward secure MACs
Forward secure MACs

0

Logger
Forward secure MACs

0 \implies 1

Logger
Forward secure MACs

0 ⇒ 1 ⇒ 2

Logger
Forward secure MACs

$\text{Logger} \rightarrow m_0 \rightarrow \text{MAC}(k_0, m_0)$
Forward secure MACs

\[
\begin{align*}
\text{Logger} & \quad m_0 \quad \text{MAC}(\text{Key}_0, m_0) \\
& \quad m_1 \quad \text{MAC}(\text{Key}_1, m_1) \\
& \quad \ldots \\
& \quad \ldots
\end{align*}
\]
Forward secure MACs

Logger

Verifier

$\text{Logger} \Rightarrow \text{Verifier} \Rightarrow 2$

$0 \Rightarrow 1 \Rightarrow 2$

$\text{m}_0$

$\text{MAC}(0, \text{m}_0)$

$\text{m}_1$

$\text{MAC}(1, \text{m}_1)$
Forward secure MACs

\[
\begin{align*}
\text{MAC}(k_0, m_0) \Rightarrow 0 \\
\text{MAC}(k_1, m_1) \Rightarrow 1 \\
\text{SipHash} \Rightarrow 2
\end{align*}
\]

Logger

Verifier

Logging to the Danger Zone: Race Condition Attacks and Defenses on System Audit Frameworks – Riccardo Paccagnella
Forward secure MACs

\[
\text{MAC}(k_0, m_0) \Rightarrow 0 \Rightarrow 1 \Rightarrow 2
\]

\[
\text{MAC}(k_1, m_1)
\]

Logger

Verifier

BLAKE2

SipHash
Forward secure MACs

Logger

Verifier

Key Precomputation

BLAKE2

SipHash

$\text{MAC}(k_0, m_0)$

$\text{MAC}(k_1, m_1)$

$m_0, m_1$
KennyLoggings - Performance

- KennyLoggings adds 340 ns to the kernel control path.
KennyLoggings - Performance

- KennyLoggings adds 340 ns to the kernel control path.
KennyLoggings - Performance

- KennyLoggings adds 340 ns to the kernel control path.
KennyLoggings - Performance

- KennyLoggings adds 340 ns to the kernel control path.

Ultimately, overhead is dictated by contention on the critical section.
Conclusion
Conclusion

• Asynchronous system logging frameworks are vulnerable to race condition attacks

Hackers are increasingly destroying logs to hide attacks

According to a new report, 72 percent of incident response specialists have came across hacks where attackers have destroyed logs to hide their tracks.
Conclusion

• Asynchronous system logging frameworks are vulnerable to race condition attacks

• Protecting against these attacks necessitates to redefine the requirements of secure logging
Conclusion

• Asynchronous system logging frameworks are vulnerable to race condition attacks

• Protecting against these attacks necessitates to redefine the requirements of secure logging

• https://bitbucket.org/sts-lab/kennyloggings